UiO : Department of Mathematics
University of Oslo

Ranking and Hillside Form

Marthe Fallang

Master's Thesis, Spring 2015

Acknowledgements

First and foremost I would like to thank my advisor, Geir Dahl, for suggesting the topic and the article that has been my main focus in this thesis. Your suggestions and ideas along the way have been very helpful.

Secondly, I am grateful to my fellow students at the study hall B601. Thank you for creating a fun environment to work in. A special thanks goes to Pia, without whom these five years at Blindern would never have been the same.

I would also like to thank my parents for their everlasting support, and for giving me a life filled with love and laughter. Thank you also for providing me with the most awesome siblings ever.

Lastly, my gratitude goes to Karl Erik, who has proofread this thesis again and again, for your patience and for all your help during the writing of this thesis.

Blindern, 26/05/2015 Marthe

Contents

Chapter 1

Introduction

The aim of this thesis is to take a closer look at the mathematics behind some ranking methods, in order to show some of the diversity in *what* one can rank and *how* one can rank these items. In addition, my goal was to take a more thorough look at the *minimum violations ranking method* found in an article written by Kathryn E. Pedings, Amy N. Langville and Yoshitsugu Yamamoto in 2010 ([PLY12]). I also wanted to use the concept of the hillside form they introduce in their article to create a ranking method of my own. Towards the end of the thesis, I interpret the hillside problem using graph theory.

Throughout the thesis we will encounter different areas of mathematics, dependent on the ranking method in question. Among these areas we mention here linear algebra, abstract algebra, optimization and graph theory.

1.1 Overview

Chapter 2 This chapter introduces the main theme of this thesis, namely ranking. We start by point out how ranking appears in many applications, both to sports and otherwise. We describe two ranking methods; Elo rating (used to rank chess players) and the Colley rating method (used to rank football teams). My presentation of these two methods follow that of Langville and Meyer in the book [LM12b]. However, the examples of Chapter 2 are my own.

Chapter 3 I look at some of the mathematics behind the ranking method PageRank, used by Google to rank web pages. My presentation follows that of chapters 2 and 4 of [LM12a]. All the examples, figures and plots are mine.

Chapter 4 This chapter is a presentation of the MVR (minimum violations ranking)

method in the article [PLY12] using my own words, and in the order I saw fit. In addition, I chose to separate what is known as Theorem 1 in the article into two results in my thesis; Lemma 4.11 and Theorem 4.12 with more detailed proofs. I also found it appropriate to formulate a corollary (Corollary 4.13) of Theorem 4.12.

The article does not give an implementation of the MVR method, so the implementation in Section 4.3 is my own. The same applies to the program that finds a decision matrix X when given a ranking π, the program that calculates the cost matrix from the point differencial matrix and also the program that calculates the number of violations from hillside form (according to definitions made in [PLY12]). All examples of Chapter 4 are my own, but in Example 4.15 we use a matrix given in the article, so we can compare my implementation's result to the result the authors got. I have made the cityplots in this chapter.

Chapter 5 The core of Chapter 5 is the hillside form, defined in the article [PLY12]. In Chapter 4 we talk of violations from hillside form, and I define the notion of distance from hillside form. Using this new notion I create a new ranking method (a minimum hillside distance ranking method) that aims to find a permutation that symmetrically reorderes a matrix to hillside form (or close to hillside form).

Section 5.2 gives a short introduction to what permutations are, following the presentation of [Fra67]. Section 5.3 is all about creating an implementation of the method, using various ideas and techniques to find a good and swift way to calculate a ranking. Sections 5.3, 5.4, 5.5 and 5.6 are my work, but in Section 5.4 I use a definition due to [DM10]. In the last section we compare the two ranking methods using hillside form on several examples, comparing the MVR method described in [PLY12] to the minimum distance ranking method created earlier in Chapter 5. I have made all the examples, with the exception of Example 5.21, which is taken from [PLY12].

Chapter 6 The final chapter of this thesis concerns more theoretical aspects of the hillside form for a matrix. I chose to look for connections here and to use graph theory to better understand what getting a matrix to hillside form means.

The chapter consists of a crash course in graph theory, some theory regarding the connection between a matrix and its adjacency graph, and also

some theory about connections between directed acyclic graphs and topological orderings. The theory of these three topics is fairly well-known, and my main sources for these topics are the first chapter of [BM76] and the lecture notes [Lee].

In Section 6.4 I connect the theory of graphs and topological orderings to matrices of hillside form, all results are original as far as I know. All examples (including drawings of graphs) in Chapter 6 are my creation. In Section 6.5 I suggest some directions for possible future work.

Appendix Here are the source code of the programs I made for this thesis.

1.2 Remarks

(i) During the work on this thesis I have written many programs. These are referenced in the text and full code can be found in the Appendix. With a few exceptions, no actual code can be found in the text itself.

(ii) This thesis concerns ranking, and we will use two different notations for a ranking of n items. Both notations will represent a vector of length n and will be denoted by $\mathbf{r}$ and $\boldsymbol{\pi}$. The different interpretations of the two notations is explained on page 44.

(iii) I will frequently end examples with the QED-square ($\square$) to make it evident when an example is finished.

(iv) Multiple places in this thesis I will speak of a *strictly upper triangular matrix*, by which I mean an upper triangular matrix with zeros on the diagonal as well as in lower triangular part of the matrix.

Chapter 2

Ranking and rating

The aim of this chapter is to give an introduction to what ranking (and rating) is, and to show some applications of ranking by presenting the Elo rating used to rank chess players and the Colley ranking used to rank (American) football teams. Both examples are borrowed from the excellent book on ranking and rating methods written by Amy N. Langville and Carl D. Meyer [LM12b].

2.1 What is ranking?

It is 7.15 a.m. and you eat breakfast while reading today's newspaper. You have reached your favourite section; the sports section. The main story is about the annoyingly brilliant Magnus Carlsen still being ranked the best chess player in the world. Another story concerns this year's Wimbledon tournament: the seedings for the traditional tennis tournament have been announced.

Now comes the local sports section. You check the table where your favorite football team is present, still holding the second to last place. With only one match left in this season it will be tough to keep the spot in your current division. Nevertheless you might climb to the safe spot that is the third to last place with a victory in the season finale.

You grab your smartphone and google for news about the team you will meet in the last match. The first result Google returns is an article about the keeper who is injured and will not play the rest of this season. Maybe your team still has a chance after all. You leave your home and go to the bus stop. It takes you to the university ranked number 185 in the world in 2013/2014[1].

[1]University of Oslo according to http://www.timeshighereducation.co.uk/world-university-rankings/2013-14/world-ranking

The morning described above contains many applications of ranking. Even though your life might not be like this, you probably encounter ranking in some form on an everyday basis. From the example above we see that ranking is not only central to sports where the use of ranking is quite obvious, but also to other areas like Google. Just as a table for teams in a fotball division, Google's PageRank algorithm returns a list of the "best" pages on the internet for your particular search phrase. The ranking method that the search engine Google uses is described in Chapter 3.

Formally, a ranking of items is a list containing the items (sorted by rank). This means that given a ranking one can say things like "item a is ranked above b", meaning that item a is listed higher than item b.

Tightly connected to ranking is rating. A rating of items gives each item a certain numerical value (a score). Thus, a sorted list of rated items is a ranking. This means that rating is a special case of ranking. A ranking of rated items contains information about *how* different two ranked items are. There are many ways to rate items, some of which are described in the next sections.

Example 2.1. At the start of every semester, the student council at the Department of Mathematics at the University of Oslo distributes study desks in Niels Henrik Abels hus. All of the mathematics students can apply, but who gets the study desks is decided by a ranking process. Let us say the student council has n study desks to distribute. After all of the applicants are ranked, the n highest ranked students get a desk each. The criteria deciding the rankings are the following:

(1) Members of the student council

(2) Master students

   (a) Group teachers

   (b) Those who had a student desk the previous semester

   (c) Progression on their degree; number of semesters/credits

(3) Bachelor students

   (a) Group teachers

   (b) Those who had a student desk the previous semester

   (c) Progression on their degree; number of semesters/credits

(4) Others

This means that all members of the student council will get study desks before the master students and among the master students those who are group teachers will get a study desk before those master students who had a study desk the previous semester, and so on. □

So why are we interested in ranking methods? Well, sometimes there is a more or less obvious way of say rewarding a winning team after a match in some sports. One could simply say that if a team won, it gets 1 point, and the losing team 0 points. At the end of the season one sorts the teams by their number of points. The team with the most points is ranked the highest and wins.

In this example the only thing of concern is the outcome of each match; who wins and who loses. For each match it does not matter at which level your opponent is, because the reward if you win is the same anyway. Neither does this simple model take into account the importance of the game, nor does it depend on how close the games were. Would it perhaps be appropriate to get more points for winning a finale in some tournament finale where the stakes are high? If yes, modifications must be made to the original ranking method.

We see that ranking methods can be made to take into account many more factors than just the win/lose factor (or yes/no in the case of Example 2.1, where your rank is determined by the "role" you possess). Thus we can improve a quite naive ranking method to one that is more complex and with more parameters.

We are also interested in ranking methods because it is not always easy to see even a simple way of ranking the items we would like to rank. Typically we must turn to other areas than sports to find such examples. This could for example be the problem of ranking the pages of the internet (as Google's PageRank does). In the case of a sports league where two and two teams meet each other there is most likely a quite simple way to rank the teams.

For ranking internet pages we do not necessarily have an obvious or naive way to do so. To speak of wins or losses gives little meaning when speaking of internet pages, so we do not even have naive way of comparing two pages. Larry Page and Sergey Brin came up with a solution to this problem when they created Google in the 90s. Chapter 3 tries to explain how they solved this problem.

Since there are many ways to both rate and rank a given set of items, the task is often to find a good rating method or a good ranking method. *Good* is quite vague and relative to whatever you compare it to, so we try to explain

what a good ranking is. A good ranking would be a ranking that represents the actual situation in a good way. For example, if a football team in a division has won all its matches in a season, one would expect that team to be ranked above the rest of the teams in the same division by the end of the season.

Of course, that ranking would mirror how good the teams are at playing and winning football matches (which is often what people find most interesting when it comes to football), but one could of course rank the teams on other merits as well. This could be things like how good the teams are at fair play for example. Possibly one could want a combination of several factors. Besides fair play, perhaps one would like to take into account how well a team plays against stronger teams and so on. How good a ranking method is therefore depends on what one wants to "measure".

We now present two solutions to two different ranking problems. The first one deals with ranking chess players, and the other one is used to rank college football teams.

2.2 Elo rating

This rating method was originally made to rate chess players, but it has later been adapted to be used in other sports as well, such as football and American football. This system has its name from the man that created it, the Hungarian-born physics professor and avid chess player Árpád Élö (1903 - 1992). He worked at Marquette University in Milwaukee, Wisconsin, and won the Wisconsin State Championship in chess eight times[2]. He created a system to rate and rank chess players (or players in other two player games), which was approved by the World Chess Federation (Federation Internationale des Echecs), FIDE, in 1970.

The main idea of the system is this: how good a chess player is or how well he performs does not change dramatically from one game to the next. What Elo proposed was that a chess player's performance is a normally distributed random variable X whose mean μ changes slowly in time (which represents the idea that a player's performance is quite stable). This sounds reasonable, as one would not expect a player who plays two players of about the same level to perform very different in the first game compared to the second game. Of course one may have good and bad games, but in Elo's model this would mean playing games where you perform to the left (worse) or to the right (better) of your supposed

[2] http://en.chessbase.com/post/arpad-emre-elo-100th-anniversary

level, your μ. This means it would take a long time for μ to change, i.e. for a player to improve his chess play. Here we present Elo's system, using the same presentation and notation as [LM12b].

Let us say that we have established some rating for a chess player. Then the only thing that can change this player's rating is the degree to which the player is performing above or below his mean, μ. We introduce the following notation. Let S be the player's recent performance/score, and let $r_{(old)}$ and $r_{(new)}$ be the player's old and new rating, respectively. Then the formula Elo came up with was this:

$$r_{(new)} = r_{(old)} + K(S - \mu),$$

where K is a constant which Elo originally set as $K = 10$. This formula is, as [LM12b] puts it, *a simple linear adjustment that is proportional to the player's deviation from his mean.* This is the classic Elo rating, but FIDE made some changes to this as it was discovered that chess performance is generally not normally distributed (as Elo proposed). The system was changed to assuming that the expected scoring difference between two players is a logistic function of the difference between their ratings. Even though this is not what Elo originally suggested, the system still carries his name and is referred to as *Elo rating.*

To explain the adjusted Elo system we introduce the following notation: let the two players be player i and player j. Continuing previous notation we let $r_{i(old)}$ and $r_{i(new)}$ be player i's old and new rating respectively. Similarly for player j. Earlier we had S, now we introduce S_{ij}:

$$S_{ij} = \begin{cases} 1 & \text{if } i \text{ beats } j \\ 0 & \text{if } j \text{ beats } i \\ \frac{1}{2} & \text{if } i \text{ and } j \text{ tie} \end{cases}$$

This corresponds to the points a player gets in a chess game; one point if he wins, none if he loses and half a point if there is a tie (a *remis*).

As mentioned earlier, the key change in this updated Elo rating method is that the expected scoring difference will be a logistic function of the difference in players' ratings. This difference in ratings we denote by

$$d_{ij} = r_{i(old)} - r_{j(old)},$$

which simply is the difference in the two players' ratings before they play. The logistic function used for chess rating is the *base-ten version* $L(x)$ of the standard logistic function $f(x) = \frac{1}{1+e^{-x}}$;

$$L(x) = \frac{1}{1 + 10^{-x}}$$

The mean μ_{ij}, which we assume is a logistic function and which should be the number of points that player i is expected to score against player j, is defined as

$$\mu_{ij} = L\left(\frac{d_{ij}}{400}\right) = \frac{1}{1+10^{-\frac{d_{ij}}{400}}}$$

Now we are ready to make the formal definition:

Definition 2.2. The Elo Rating Formulas used to rate and rank chess players are the following:

$$r_{i(new)} = r_{i(old)} + K(S_{ij} - \mu_{ij}) \quad \text{and} \quad r_{j(new)} = r_{j(old)} + K(S_{ij} - \mu_{ij})$$

where S_{ij} and μ_{ij} are defined as

$$S_{ij} = \begin{cases} 1 & \text{if } i \text{ beats } j \\ 0 & \text{if } j \text{ beats } i \\ \frac{1}{2} & \text{if } i \text{ and } j \text{ tie} \end{cases} \quad \text{and} \quad \mu_{ij} = \frac{1}{1+10^{-\frac{d_{ij}}{400}}}$$

To most people it would be seen as a greater achievement for a weak player to beat a strong player than vice versa. Most likely we would expect the stronger player to win. In terms of change in rating, this means that if the stronger player wins we would like him to only get a small increase in his rating. But if, contrary to our expectations, the weaker player was to win, we would like to reward this player with a bigger increase in his rating that what the stronger would have gotten if he had won. Luckily Elo's system agrees with this intuition, as Example 2.3 verifies.

Example 2.3. Magnus Carlsen's FIDE-rating is 2881 (6th of March 2014), the highest FIDE-rating ever. Let us say that he is to play a player which has a rating of 1700. Now we can apply the Elo rating formulas to see what the outcome would be, in terms of rating changes, if these two chess players met in a match. We use the abbreviations *mc* for Magnus Carlsen and *pl* for the other player. Now we use the formulas from Definition 2.2:

$$\mu_{pl,mc} = \frac{1}{1+10^{-\frac{(1700-2881)}{400}}} \approx 0.0011$$

$$\mu_{mc,pl} = \frac{1}{1+10^{-\frac{(2881-1700)}{400}}} \approx 0.9989$$

Recall that μ_{ij} is the number of points that we expect player i to score against player j. So in this example we would expect Magnus Carlsen to score 0.9989 points against the other player. Since the maximum amount of points one can

get is 1, this means we are quite sure that Magnus Carlsen will win this match. But what happens to the two players' ratings after their game? Recall that if player i beats player j, then $S_{ij} = 1$. If the player with 1700 in FIDE-rating wins we have the following situation:

$$r_{pl(new)} - r_{pl(old)} = K(S_{pl,mc} - \mu_{pl,mc}) = K(1 - 0.0011) = 0.9989K$$

And if Magnus Carlsen wins we have:

$$r_{mc(new)} - r_{mc(old)} = K(S_{mc,pl} - \mu_{mc,pl}) = K(0.0011 - 1) = 0.0011K$$

We see that the reward the weaker player gets if he (unexpectedly) beats Magnus Carlsen if far bigger than if Magnus Carlsen should win, as this is what is expected of him. Exactly how many points they get is controlled by the factor K. □

The factor K plays a central role in the Elo rating formulas. As we saw in Example 2.3 K determines the increase or decrease in a player's rating after a match. We can say that K determines the pace at which ratings changes. This means that if we choose K too small, the reward is small even if one beats a superior opponent.

However, if K is chosen too big, then the opposite is the case. Too much weight would be put to even the smallest deviation from ones established performance level. For example, playing just below expectations could result in a severe decrease in ones rating. Originally, Elo chose $K = 10$, but one can change this factor according to for example the importance of the game or the level of competition.

Example 2.4. The K factor that FIDE uses today, changes with the level of competition. These changes are described on FIDE's webpages[3], and are summarized here:

(1) $K = 30$ for a player until he or she has completed at least 30 rated matches.

(2) $K = 15$ for a player with a rating that has never exceeded 2400.

(3) $K = 10$ for a player with over 30 rated matches and with a rating that has reached 2400 at some point. After this K remains at 10. □

[3] http://en.chessbase.com/post/fide-april-2013-ratings--and-reform-plans-050413

2.3 The Colley rating method

The method we are going to present now was developed by Dr. Wesley Colley. Colley has a Ph.D. in astrophysics from Princeton, but published a paper in the early 2000s on a method for ranking sports teams ([Col02]). One way of telling the success of the Colley Rating Method (or just Colley's method) is the fact that it is incorporated in the BCS (the Bowl Championship Series) method of ranking NCAA (National Collegiate Athletic Association) college football teams.

To explain Colley's method we first introduce some notation, in addition to the simple rating system that Colley build his own method on. This original rating system uses the notion of *winning percentage*, which assigns team i to the value r_i determined by

$$r_i = \frac{w_i}{t_i}$$

Here w_i is the number of wins for team i and t_i the total number of games played by team i. This quite simple rating system is used in many leagues and tournaments around the world, both recreational and professional ones.

So why would Colley try to improve this seemingly usable rating method? It is because this original method has some weaknesses. Let us say that one uses this method to rank the football teams in a league one season. At season's end, everyone is wondering which team came first. Since we have the ratings r_i for every team, this is just a question of sort the teams from highest to lowest rating, that is get the ranking based on the ratings. But since (most likely) all teams have played the same number of games, ties in the ratings may very well occur. This method can therefore lead to ties in the ratings, which is unfortunate because it becomes harder to rank the teams in the end.

Another weakness of this method is the fact that it does not take into account what kind of opponent your team is facing, that is if your opponent is a strong or weak team. The reward for beating a strong team is the same as the reward for beating a weak team. This is unlike for example the Elo rating of chess players we saw in Section 2.2, where the strength of your opponent is a crucial part of determining your new rating. The last weakness we will point out is that in the beginning of each season, when no games have been played, the winning percentage r_i of each team is $r_i = \frac{0}{0}$. This also leads to that a team that never wins has rating 0. These problems Colley tried to fix with his new method, presented below as in [LM12b]:

Definition 2.5. The Colley method begins with a slight modification to the traditional winning percentage formula so that

$$r_i = \frac{1 + w_i}{2 + t_i}$$

We notice immediately that this adjustment[4] gets rid of the $\frac{0}{0}$ problem in the beginnning of each season, neither do we have the rating 0 for a team without victories. Now, at the beginning of each season, each team starts with rating $r_i = \frac{1}{2}$.

Colley's ranking method can be expressed as a system of linear equalities. The details on how Colley took Definition 2.5 to a matrix formulation can be found in his article [Col02]. Below we merely summarize his method.

Definition 2.6. The Colley Rating Method can be summarized as follows, using matrix notation.

$\boldsymbol{C}_{n\times n}$ real symmetric positive definite matrix called the *Colley matrix*;

$$\boldsymbol{C}_{ij} = \begin{cases} 2 + t_i & i = j \\ -n_{ij} & i \neq j \end{cases}$$

t_i total number of games played by team i

n_{ij} number of times teams i and j faced each other

$\boldsymbol{b}_{n\times 1}$ right-hand side vector; $b_i = 1 + \frac{1}{2}(w_i - l_i)$

w_i total number of wins accumulated by team i

l_i total number of losses accumulated by team i

$\boldsymbol{r}_{n\times 1}$ general rating vector produced by the Colley system

n number of teams in the league

A rating vector $\boldsymbol{r}$ containing the Colley ratings can be obtained by solving the system $\boldsymbol{C}\boldsymbol{r} = \boldsymbol{b}$.

[4]This adjustment comes from what is known as *Laplace's rule of succession.*

We see that the Colley matrix's entries only depend on the total number of games played by each team (the diagonal elements of $\boldsymbol{C}$) and on the number of times each pair of teams has played each other (the non-diagonal elements of $\boldsymbol{C}$).

This means that the Colley matrix itself does not contain any information about the outcome any particular match. The win-loss information can be found in the vector $\boldsymbol{b}$, but even here it is only the number of wins and the number of losses for each team i that plays a role. It is not possible to get out any information about one specific result between team i and team j; not about which team won and not about the actual scoring in the match. These properties make sure that the results of the Colley method are *bias-free*, as Colley himself puts it.

Example 2.7. We illustrate Colley's method. Let us say we have a four-team handball division where each team meet each other exactly one time (this could be for example the group stage in a tournament). Let us call these teams a, b, c and d. The results are these:

Table 2.1: Results

	a	b	c	d	# victories	# losses
a		12-23	20-18	23-25	1	2
b	23-12		27-11	26-23	3	0
c	18-20	11-27		16 -25	0	3
d	25-23	23-26	25-16		2	1

According to the definitions made in Defintion 2.6 these results give the following Colley system:

$$\boldsymbol{C}\boldsymbol{r} = \begin{pmatrix} 5 & -1 & -1 & -1 \\ -1 & 5 & -1 & -1 \\ -1 & -1 & 5 & -1 \\ -1 & -1 & -1 & 5 \end{pmatrix} \begin{pmatrix} r_a \\ r_b \\ r_c \\ r_d \end{pmatrix} = \begin{pmatrix} 0.5 \\ 2.5 \\ -0.5 \\ 1.5 \end{pmatrix} = \boldsymbol{b}$$

We let MATLAB solve this system, by writing `r = C\b`. The resulting Colley rating vector is

$$\boldsymbol{r} = \begin{pmatrix} r_a \\ r_b \\ r_c \\ r_d \end{pmatrix} = \begin{pmatrix} 0.42 \\ 0.75 \\ 0.25 \\ 0.58 \end{pmatrix}$$

Now, to obtain the ranking of the four teams, we sort each of the teams after descending ratings. This gives us r_b, r_d, r_a, r_c. This means that highest ranked

team is team b, team d is in second place and so on. We know from Table 2.1 that team b won all its matches, so it seems fair that it is ranked highest. □

2.4 More on ranking methods

Now we have seen examples of solutions to a couple of ranking problems. Even though we have only seen two, we can already understand that there are a lot of different ways of handling a ranking problem. Which solution one should choose or try to find depends on which factors one would like to take into account. Even though the Elo rating method is very well suited for rating chess players, it may not be as good at rating football teams in a league as the Colley method is. The methods we have discussed so far are well suited for their main application (in our cases, chess and football) because they are tailored to suit exactly these applications.

Of course the methods we have seen so far could be more refined, could have handled something better and could have considered more aspects of the match etc. For example, neither Elo nor Colley take into account the importance of a match. In the case of Elo's rating method this issue could be solved by raising the K-factor. However, there are other methods that have this importance weighting as a central part of the ranking method. This is the case in Geir Dahl's article [Dah12] on a matrix method for ranking tennis players. Here, every match m is given a weight $\beta_m \geq 0$. This weight is a measure of the importance of the match m.

We have now seen some solutions to different ranking problems, mainly for applications to sports. In the next chapter we will take a look at a very different ranking problem, namely the problem of ranking pages of the internet. There are of course many things besides webpages and sports teams that can be ranked. For example the ranking of academic journals. An academic journal is ranked on the basis of severeal things, among these both the quality and the impact of the journal. Finally we also mention the many international rankings[5] that exist, where countries are ranked in many different categories, from literacy rate to health care quality. None of these rankings will be of any concern to us in this thesis. We shall focus on webpages and sports.

[5] http://en.wikipedia.org/wiki/List_of_international_rankings

Chapter 3

Google's PageRank

The aim of this chapter is to give an overview of the ranking method behind the search engine Google. We present some of the mathematics, in particular the linear algebra, that is at the core of this very successful search engine. The main sources for this part of the thesis is the book *Google's PageRank and Beyond* ([LM12a]) by Amy N. Langville and Carl D. Meyer, in addition to the artice *The $25,000,000,000 Eigenvector: The Linear Algebra behind Google* ([BL06]) by Kurt Bryan and Tanya Leise.

3.1 Ranking webpages

Let us say one were given a web, i.e. a set of webpages. If one was to rank these pages after relevance relative to a search phrase, how should one proceed? This is, in essence, the problem of ranking webpages.

The process of ranking webpages consists of three phases, roughly and quite simplified speaking. These three are *crawling*, *indexing* and *ranking*. The first two of these, namely the crawling and the indexing, happens before any queries have been submitted, only the ranking part of the process happens after a search phrase has been submitted.

The first thing one must do when one wants to rank a web, is to get some spiders (of course). The spiders are present in the crawling phase of the webpage ranking, and are essentially virtual robots wandering around the web searching for new webpages and new information. The spiders gather the new webpages in a central repository until they are indexed by the indexing module. The indexing module extracts only the most vital information of the webpage, thus creating a compressed description of the webpage. This compressed description is stored in

various indexes, to be accessed when needed later in the ranking process. The indexes and how they are created and structured is rather involved, for our purpose it is enough to know that these indexes exist and that they contain compressed information about webpages.

Let us assume that we have the indexes described above. If a search phrase is emitted to Google, what is called the query module comes into play. The query module takes the search phrase and converts it to a form that can be used to look the phrase up in the indexes. The query module then returns a list of what is called relevant pages.

These pages are then given to the last instance of the process, namely the ranking module, which returns an ordered list of webpages. It is the ranking method that Google uses in this part of the ranking process, the PageRank, that will be of interest to us for the rest of this chapter.

3.2 PageRank

Now we will briefly present the ranking method known as PageRank, invented by Larry Page and Sergey Brin in the late 1990s. We follow the presentation of Langville and Meyer do it in [LM12a].

Often when one wants to find a ranking of some elements, one first finds a more or less good way to rate the same elements. This means to assign a value to each element such that if page A has higher rating than page B, page A is better than page B in some sense. One then sort the elements by their given rating, and this gives the ranking.

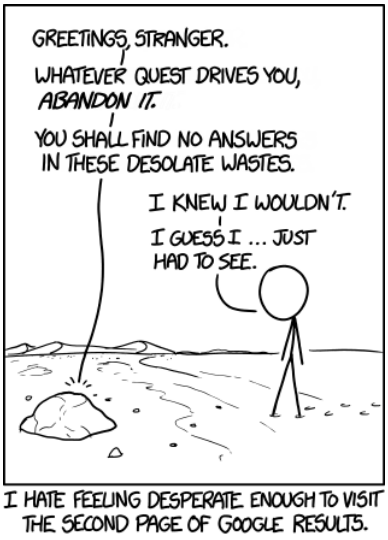


Figure 3.1: An xkcd panel illustrating how good PageRank is.[0]

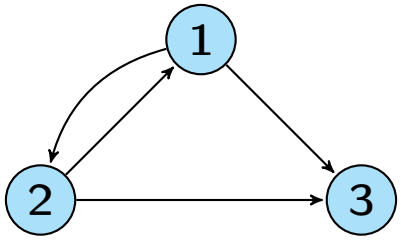
However, how would one proceed to rate a webpage? What makes a webpage deserving of a higher rating than another webpage? Brin and Page answered these questions with *importance*. How important your webpage is should determine the webpage's rating. The importance is measured by how many other webpages that link to one particular webpage. Surely, if many different webpages contain

[0]Property of Randall Munroe, https://xkcd.com/1334/.

links to webpage A, then webpage A must be important. The word *backlinks* is used for the links to a specified page.

Given what we know of the web and backlinks, we may represent the web as a directed graph[1]. We let the nodes be the n pages that the web consists of, and let the edges be given as the backlinks.

Example 3.1. Let us say we have tiny web, consisting of $n = 3$ webpages. Then we could assign each page to a node in a graph and let the edges be the backlinks between the pages. We can illustrate this as below.



Here we see that P_1 links to both P_2 and P_3. P_3 contains no links to either of the other pages in this web. □

Using the same notation as in [LM12a] we let the PageRank of page P_i be denoted by $r(P_i)$. Furthermore, let B_{Pi} be the set of pages pointing to P_i and $|P_j|$ the number of outlinks from page P_j. The initial PageRank formula that Page and Brin suggested was

$$r(P_i) = \sum_{P_j \in B_{Pi}} \frac{r(P_j)}{|P_j|} \tag{3.1}$$

We see that the PageRank of page i, P_i, depends on the PageRanks of all pages P_j that links to page i. This captures Page and Brin's idea of importance. How good your page is is heavily dependent on how important the pages that point to your page is (i.e. how "much" PageRank they have). The PageRank for page j, $r(P_j)$, is scaled as we see in Equation 3.1 by $|P_j|$, i.e. the number of pages that P_j links to. This means that it is better to be one of few pages pointed to by an important page, than to be one of very many pages pointed to by an important page.

Equation 3.1 is a seemingly good way to compute the PageRank of a page. However, one immeadiate problem occurs: $r(P_j)$ is unknown for all pages j! Brin

[1]For formal definitions regarding graph theory see Section 6.1, page 67.

and Page solved this problem by letting the problem of finding PageRank become an iterative procedure. If we assume that the web consists of n pages, then the initial situation Brin and Page suggested was $r_0(P_i) = \frac{1}{n}$ for $i = 1, \ldots, n$. After the kth iteration the PageRank of page i is given as

$$r_{k+1}(P_i) = \sum_{P_j \in B_{Pi}} \frac{r_k(P_j)}{|P_j|} \tag{3.2}$$

Example 3.2. Now we can calculate some iterations for our small web graph in Example 3.1. Using the definitions we find that the initial PageRank for each of the three pages is $\frac{1}{3}$. Then we use Equation 3.2 to see what the PageRanks are for $k = 1$, i.e. after one iteration:

$$r_1(P_1) = \sum_{P_j \in B_{P_1}} \frac{r_0(P_j)}{|P_j|} = \frac{r_0(P_2)}{|P_2|} = \frac{\frac{1}{3}}{2} = \frac{1}{6}$$

$$r_1(P_2) = \sum_{P_j \in B_{P_2}} \frac{r_0(P_j)}{|P_j|} = \frac{r_0(P_1)}{|P_1|} = \frac{\frac{1}{3}}{2} = \frac{1}{6}$$

$$r_1(P_3) = \sum_{P_j \in B_{P_3}} \frac{r_0(P_j)}{|P_j|} = \frac{r_0(P_1)}{|P_1|} + \frac{r_0(P_2)}{|P_2|} = \frac{\frac{1}{3}}{2} + \frac{\frac{1}{3}}{2} = \frac{1}{3}$$

□

Now that Brin and Page had the iterative procedure (Equation 3.2) one might think that they were satisfied, but no. Being mathematicians they saw that they could reformulate Equation 3.2 using matrices. Instead of computing the PageRank, $r_k(P_i)$, for one page at the time, one can compute a $1 \times n$ vector (assuming it is n pages in the web) holding the PageRank for all the webpages at once. To transform Equation 3.2 to a matrix problem, Brin and Page came up with the $n \times n$ matrix $\boldsymbol{H}$ and the $1 \times n$ row vector $\boldsymbol{\pi}^{(k)T}$, which is the PageRank vector after the kth iteration. The PageRank for page i will then be found at $\pi_i^{(k)T}$ (the ith element of of the PageRank vector). The entries in the matrix $\boldsymbol{H}$ is given by

$$H_{ij} = \begin{cases} \frac{1}{|P_i|} & \text{if there is a link from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases}$$

As before, $|P_i|$ is the number of outlinks from P_i. Brin and Page then formulated Equation 3.2 as

$$\boldsymbol{\pi}^{(k+1)T} = \boldsymbol{\pi}^{(k)T}\boldsymbol{H} \tag{3.3}$$

With the initial PageRank vector $\boldsymbol{\pi}^{(0)T} = \frac{1}{n}\boldsymbol{e}^T$, where $\boldsymbol{e}^T$ is a row vector with 1 in all entries.

Example 3.3. Once again we return to the web in Example 3.1. Using the new matrix and vector formulations we get the following for the initial PageRank vector $\boldsymbol{\pi}^{(0)T}$ and the matrix $\boldsymbol{H}$.

$$\boldsymbol{\pi}^{(0)T} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \end{pmatrix}$$

$$\boldsymbol{H} = \begin{pmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \\ 0 & 0 & 0 \end{pmatrix}$$

Now we can verify that the matrix formulation 3.3 yields the same result as Equation 3.2 after one iteration:

$$\boldsymbol{\pi}^{(1)T} = \boldsymbol{\pi}^{(0)T}\boldsymbol{H} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \end{pmatrix} \begin{pmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1/6 & 1/6 & 1/3 \end{pmatrix}$$

We see that the PageRank for each page is the same as in Example 3.2. □

Brin and Page still had some way to go to get to the PageRank formulation they could actually use. Some problems became evident when they used Equation 3.3. Two of these problems, *cycles* and *rank sinks*, are described in Examples 3.4 and 3.5.

Example 3.4. The graph in Example 3.1 contains what is called a *rank sink*. We see that page 3, P_3, contains no outgoing links, but both P_1 and P_2 links to P_3. This will lead to P_3 getting more and more rank, without ever "giving" rank, as the PageRank method iterates. We call a node without outgoing links a *dangling node*. Thus, P_3 is a dangling node in our case. In Examples 3.2 and 3.3 we saw that after one iteration P_3 was the page with highest PageRank. □

Example 3.5. Another problem is the so called *cycles*. If we remove P_3 from our small web graph example we are left with only P_1 and P_2, both containing links to each other.

This situation would lead to that the two pages would "have" PageRank every other time, depending on what the initial situation was. In other words we would have an infinite loop, and it would be very hard to conclude which webpage is the better one, i.e. the method does not converge for this case.

□

In Example 3.3 we noticed that a dangling node in the web graph resulted in a 0 row in the matrix $\boldsymbol{H}$. However, in our example the nondangling nodes, i.e. P_1 and P_2, resulted in stochastic rows[2]. Thus $\boldsymbol{H}$ is what is called *substochastic*. Furthermore, in [LM12a], Langville and Meyer remark that $\boldsymbol{H}$, in general, looks very much like a stochastic transition probability matrix for a Markov chain[3]. Because the Markov theory is well explored, and if $\boldsymbol{H}$ had been a Markov matrix we would have known what properties it ought to have for Equation 3.3 to converge to a unique positive PageRank vector $\boldsymbol{\pi}^{(k)T}$.

Therefore we wish that $\boldsymbol{H}$ was a stochastic, irreducible and aperiodic matrix[4]. This would ensure convergence to a unique PageRank vector (no matter what the initial vector $\boldsymbol{\pi}^{(0)T}$ is). Even though Brin and Page do not mention Markov chains in their original paper, they went on and made adjustments to Equation 3.3 according to the properties above (stochasticity etc.). However, instead of speaking explicitly of Markov theory they introduced a *random surfer* to their model.

Brin and Page's random surfer sits in front of his computer, surfing around the web, clicking on links. But what if the surfer enters a webpage and finds no links to click on (he has found one of the web's dangling nodes.)? Well, he may of course enter a new web address in the URL line and surf on the new page instead, and then click on some more links and then jump to any page again. This activity can be described in mathematical terms as well, and it was this behavior that led to what Langville and Meyer call the stochasticity adjustment and the primitivity adjustment. Since this is described very well in [LM12a] we just sum up the changes here.

The stochasticity adjustment solves the rank sink problem we saw in Example 3.4. It makes sure that the dangling nodes no longer contributes with 0 rows in the matrix $\boldsymbol{H}$. The 0 rows are instead replaced with $\frac{1}{n}\mathbf{e}^T$ (again every entry in $\mathbf{e}$ is 1). Since $\sum_{i=1}^{n} \frac{1}{n} 1 = \frac{n}{n} = 1$, this adjustment makes $\boldsymbol{H}$ stochastic. This illustrates that a random surfer can enter any of the nodes in the web (even after entering a dangling node). To write this adjustment using our matrix $\boldsymbol{H}$ a new (and stochastic) matrix is introduced:

$$\boldsymbol{S} = \boldsymbol{H} + \mathbf{a}\left(\frac{1}{n}\mathbf{e}^T\right)$$

[2]A stochastic row is a nonnegative, real row where the elements in the row sum to 1.
[3]For definitions and more on this subject we refer to pages 687-702 in [Mey00].
[4]See footnote 3.

where the entries in the *dangling node vector* $\mathbf{a}$ is defined as

$$a_i = \begin{cases} 1 & \text{if node } i \text{ is a dangling node} \\ 0 & \text{otherwise} \end{cases}$$

The second adjustment, the primitivity adjustment, reflects the behavior of a random surfer when he does both some link-clicking and some actual typing of a new web address. Brin and Page therefore introduced the constant α, $\alpha \in [0, 1]$, which is how much of the total surf time the random surfer spends on mere link-clicking. Then, since we only have defined two types of surf actions (link-clicking and jumping to any page by writing the explicit address in the URL line), $(1-\alpha)$ will be the proportion of the total surf time the surfer uses on writing a new address and not on following the link-structure of the web.

To preserve the newly achieved stochasticity of the matrix $\boldsymbol{S}$ Brin and Page introduce yet another matrix, $\boldsymbol{G}$, which is a convex combination of two stochastic matrices. These two matrices are the two types of actions the random surfer can do, namely link-clicking or jump to a random page. The matrix $\boldsymbol{G}$, named the *Google matrix*, is defined as

$$\boldsymbol{G} = \alpha \boldsymbol{S} + (1 - \alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T$$

where α and $\boldsymbol{S}$ are described as earlier. The matrix $\frac{1}{n}\mathbf{e}\mathbf{e}^T$ is stochastic and captures that when the random surfer jumps to any page, it is random what page the surfer jumps to.

These two adjustments gives $\boldsymbol{G}$ the desired properties, so that the Markov theory can be applied. Thus the adjusted method for computing the Google PageRank is the following

$$\boldsymbol{\pi}^{(k+1)T} = \boldsymbol{\pi}^{(k)T}\boldsymbol{G} \tag{3.4}$$

This concludes the discussion of the PageRank method in this thesis. For more on this subject, see the excellent book [LM12a].

Chapter 4

A minimum violations ranking method

This chapter will concern the article *A minimum violations ranking method* ([PLY12]) written by Kathryn E. Pedings, Amy N. Langville and Yoshitsugu Yamamoto. Here we will present their method and ideas, and fill in some details here and there, in addition to examples to illustrate their concepts.

4.1 Hillside form and ranking

Here we will try to motivate and build a method for finding a ranking of n items that have been compared pairwise. We will rank these items on data given in a point differential matrix.

Definition 4.1. A *point differential matrix* $\boldsymbol{D}$ is an $n \times n$ matrix which contains results of pairwise comparisons of n elements. If the n elements are sports teams, then each element in a point differential matrix $\boldsymbol{D}$ may be the difference in goals when two teams faced each other in a match. If team i loses to team j $d_{ij} = 0$.

Example 4.2. We will now see what information a point differential matrix holds. Assume $\boldsymbol{D}$ below is a point differential matrix for four teams in some sport (for example football).

$$\boldsymbol{D} = \begin{pmatrix} 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 3 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Since the entry $(1, 2)$ is 2, this means that team 1 beat team 2 by two points or goals in their match. Furthermore, $(2, 1)$ must be zero, as team 2 lost to team 1.

We can also see that team 3 won all its three matches. Against team 1 it won by 2 points, against team 2 with 3 points and against team 4 with 4 points.

□

A central concept in the article [PLY12] is the notion of hillside form for a square matrix. The formal definition is the following:

Definition 4.3. An $n \times n$-matrix $\boldsymbol{D} = [d_{ij}]_{i,j=1}^{n}$ is in *hillside form* if

$$\begin{aligned} d_{ij} &\leq d_{ik} \quad \forall i \text{ and } \forall j \leq k \quad \text{(ascending order across rows)} \\ d_{ij} &\geq d_{kj} \quad \forall j \text{ and } \forall i \leq k \quad \text{(descending order down columns)} \end{aligned}$$

Said differently this means that the largest element in a matrix on hillside form is found in the uppermost right corner of the matrix, and that any given element in the matrix will have larger (or equal) elements above and to the right, and smaller (or equal) elements below and to the left.

A matrix in hillside form is a dream situation. In terms of point differentials it means that team 1 has beaten all the other teams, and in addition it has beaten the worst teams by more points than those which are just a little worse than itself. The same applies for the second best team; the only team they lost to is the one ranked first, and team 2 beat team 3 by less points than it beat team 4 by and so on.

However, for matrices that are *almost* in this dream situation, Pedings, Langville and Yamamoto introduce two notions to describe the entries that takes the matrix away from the dream situation of hillside form; *weak wins* and *upsets*. Weak wins are entries that breaks the hillside pattern in the upper triangular part of the matrix.

Upsets are entries that violates the hillside pattern in the lower triangular part of the matrix. The upset can easily be spotted as they are any nonzero entries in the lower triangular part of the matrix. The names of these notions are of course not arbitrary: a weak win is whenever a team does not beat a lower-ranked team by the amount of points that is expected of it, given the differences in rankings between the two teams. Upsets occur whenever a team looses to a lower-ranked team.

Example 4.4.

$$\boldsymbol{A} = \begin{pmatrix} 0 & 7 & 3 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \boldsymbol{B} = \begin{pmatrix} 0 & 7 & 0 \\ 0 & 0 & 3 \\ 1 & 0 & 0 \end{pmatrix}$$

In $\boldsymbol{A}$ the entry 3 is a weak win, and in $\boldsymbol{B}$ the entry 1 is an upset. □

Since the matrices we will concern ourselves with are based on pairwise comparisons, the diagonal of these matrices will be zero (as we can hardly give any other meaning to comparing something to itself). We could therefore have added in Definition 5.1 a line about the matrices in hillside also needing to be strictly upper triangular (i.e. $d_{ii} = 0$, for $1 \leq i \leq n$). However these zeros will occur naturally when we assume that something compared to itself is represented by a zero. Now, let us see some examples of matrices in hillside form.

Example 4.5.

$$\boldsymbol{A} = \begin{pmatrix} 0 & 3 & 4 & 9 & 17 \\ 0 & 0 & 2 & 8 & 12 \\ 0 & 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 0 & 1 & 7 & 5 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad \boldsymbol{C} = \begin{pmatrix} 0 & 6 & 10 \\ 0 & 0 & 9 \\ 0 & 0 & 0 \end{pmatrix}$$

$\boldsymbol{A}$ and $\boldsymbol{C}$ are matrices in hillside form, while $\boldsymbol{B}$ is not in hillside form. □

Next we introduce the titular *violations* from hillside form. In the matrix $\boldsymbol{B}$ in Example 4.5, the entry 5 is misplaced for the matrix to be in hillside form (it represents a weak win). If we look only at column number three, there is nothing wrong in terms of the hillside definition. But when we look at row 1, we see that something is wrong. We do not have an ascending order across this row, so we have a violation of the hillside definition.

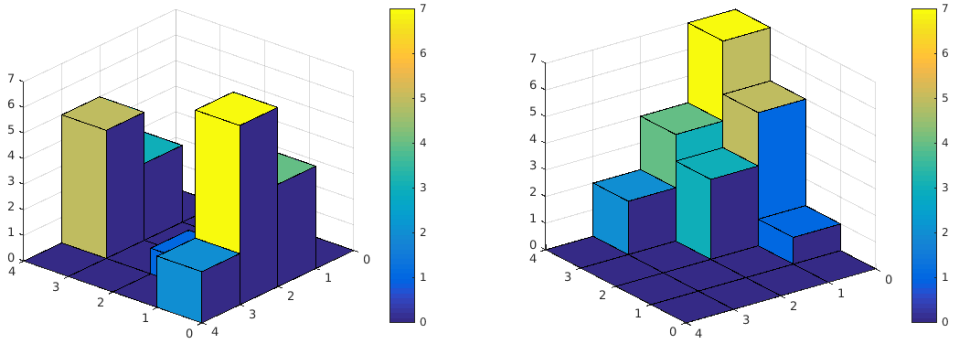
This means that $\boldsymbol{B}$ has one violation ($7 > 5$). In this fashion one could count violations for any matrix. One just has to check violations both for columns and rows. Any entry in a matrix has several "chances" of being a violation.

So why are we interested in the hillside concept when we talk of ranking? This is due to the fact that for an $n \times n$-matrix in hillside form, the ranking of the n items is quite clear. If we look at the matrix $\boldsymbol{A}$ in Example 4.5, we see that there is only one reasonable ranking of its 5 elements. Namely the ranking $\boldsymbol{r} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix}$. Since team 1 beat all the other teams, team 2 beat all teams except for team 1, and so on, this appears to be a reasonable ranking of the 5 items. In this example our matrix already was in hillside form and we can read the ranking directly, but what about a matrix that is not in hillside form?

Example 4.6. Assume we have the following two matrices.

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 7 & 1 & 0 & 5 \\ 2 & 0 & 0 & 0 \end{pmatrix}, \quad \boldsymbol{D} = \begin{pmatrix} 0 & 1 & 5 & 7 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

We see readily that the matrix $\boldsymbol{D}$ is in hillside form, but that the matrix $\boldsymbol{A}$ is not so fortunate. Nevertheless, these two matrices share a connection. Indeed, if $\boldsymbol{A}$ is symmetrically reordered[1] according to the vector $\begin{pmatrix}3 & 2 & 4 & 1\end{pmatrix}$, the result is the hillside matrix $\boldsymbol{D}$.



For this example we also show the corresponding cityplots of $\boldsymbol{A}$ (left) and $\boldsymbol{D}$ (right), illustrating why the term *hillside* is fitting. These cityplots are made with MATLAB. □

In example 4.6 we saw that it is possible to symmetrically reorder $\boldsymbol{A}$ to hillside form. We will sometimes talk of matrices having a *hidden hillside form*. Such matrices are not in hillside form originally, but can be symmetrically reordered to a hillside matrix. Of course not all matrices have an underlying hillside form, but it might be possible to take a matrix closer to hillside form by finding reorderings as in Example 4.6. By closer to hillside form in this setting we mean less violations than the previous one.

To find such a hidden hillside form (or as close as it gets) for a matrix is the aim of the minimum violation ranking method (MVR method) of Pedings et. al. Given a point differential matrix, their method tries to find reorderings such that the reordered matrix is in hillside form or close to it. From this reordered matrix they read their ranking. This MVR problem may be stated mathematically as follows.

[1]This really means that both the columns and the rows are reordered according to the same vector. In other words, the columns and rows are permuted. One can read more about this and permutation matrices in Section 5.2, where we treat this topic in more detail than they do in [PLY12].

Problem 4.7. Given a point differential matrix $\boldsymbol{D}$ of dimension $n \times n$, we want to find an $n \times n$ permutation matrix $\boldsymbol{Q}$ so that the symmetrically reordered matrix $\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q}$ has minimal hillside violations.

$$\begin{aligned} \min_{\boldsymbol{Q}} \ & \#\text{ hillside violations of } \boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q} \\ \text{s.t. } & \boldsymbol{Q}^T\boldsymbol{e} = \boldsymbol{e} \\ & \boldsymbol{e}^T\boldsymbol{Q} = \boldsymbol{e}^T \\ & q_{ij} \in \{0, 1\} \end{aligned}$$

Where $\boldsymbol{e}$ is the vector of length n with all entries equal to 1.

This formulation belongs to the class of quadratic integer programs (QIPs), and unfortunately these problems are rather challenging to solve. However, the next step towards the MVR method is to rewrite this problem to a more solvable problem.

4.2 BILP formulation

Now we will try to take the QIP formulation in Problem 4.7 and reformulate it to a binary integer linear program (BILP). Hopefully this will be easier to solve than the QIP formulation. To reformulate the problem as they do in [PLY12] we will need some more constructions and definitions. We start with the following definition of the cost matrix $\boldsymbol{C}$.

Definition 4.8. Let $\boldsymbol{C} = [c_{ij}]$ $\forall i, j = 1, 2, \ldots, n$ be an $n \times n$-matrix defined as

$$c_{ij} = \#\{k \mid d_{ik} < d_{jk}\} + \#\{k \mid d_{ki} > d_{kj}\}$$

where $\#$ denotes the cardinality.

Before we continue and show that the matrix $\boldsymbol{C}$ can be used to compute the number of violations to hillside form, we will try to give some intuition to what the matrix $\boldsymbol{C}$ reflects. First, observe that the number $\#\{k \mid d_{ik} < d_j\}$ is the number of teams receiving a lower point differential against team i than team j (if one looks at how we defined the point differential matrix $\boldsymbol{D}$, one see that this interpretation of the set $\{k \mid d_{ik} < d_j\}$ seems reasonable). Likewise, we interpret $\#\{k \mid d_{ki} > d_{kj}\}$ as the number of teams receiving a greater point differential against team i than team j. We also notice that the diagonal of $\boldsymbol{C}$ will only consist of zeros, this is because when $i = j$ we have that

$$c_{ii} = \#\{k \mid d_{ik} < d_{ik}\} + \#\{k \mid d_{ki} > d_{ki}\} = 0$$

Now, let us turn to an example for the $n = 3$ case and see if we might get some more insight as to what $\boldsymbol{C}$ look like.

Example 4.9. Say we have three teams, called team 1, 2 and 3, respectively. These teams all met each other once in a tournament. The results of the matches were these:

$$\begin{aligned}&\text{Team 1 against Team 2: } 14 - 23\\ &\text{Team 1 against Team 3: } 18 - 21\\ &\text{Team 2 against Team 3: } 17 - 15\end{aligned}$$

What is the corresponding point differential matrix $\boldsymbol{D}$? Well, we must have that the $(2,1)$-entry is $23-14 = 9$, the $(3,1)$-entry is $21-18 = 3$, and the $(2,3)$-entry must be $17 - 15 = 2$. The rest of the entries are zero. Thus what we get is this:

$$\boldsymbol{D} = \begin{pmatrix} 0 & 0 & 0 \\ 9 & 0 & 2 \\ 3 & 0 & 0 \end{pmatrix}$$

Now we turn to the cost matrix $\boldsymbol{C}$. We check what each entry in the matrix must be, according to Definition 4.8.

$$\begin{aligned}
c_{11} &= \#\{k \mid d_{1k} < d_{1k}\} + \#\{k \mid d_{k1} > d_{k1}\} = 0\\
c_{12} &= \#\{k \mid d_{1k} < d_{2k}\} + \#\{k \mid d_{k1} > d_{k2}\} = \#\{k = 1,3\} + \#\{k = 2,3\} = 4\\
c_{13} &= \#\{k \mid d_{1k} < d_{3k}\} + \#\{k \mid d_{k1} > d_{k3}\} = \#\{k = 1\} + \#\{k = 2,3\} = 3\\
c_{21} &= \#\{k \mid d_{2k} < d_{1k}\} + \#\{k \mid d_{k2} > d_{k1}\} = 0\\
c_{22} &= \#\{k \mid d_{2k} < d_{2k}\} + \#\{k \mid d_{k2} > d_{k2}\} = 0\\
c_{23} &= \#\{k \mid d_{2k} < d_{3k}\} + \#\{k \mid d_{k2} > d_{k3}\} = 0\\
c_{31} &= \#\{k \mid d_{3k} < d_{1k}\} + \#\{k \mid d_{k3} > d_{k1}\} = 0\\
c_{13} &= \#\{k \mid d_{3k} < d_{2k}\} + \#\{k \mid d_{k3} > d_{k2}\} = \#\{k = 1,3\} + \#\{k = 2\} = 3\\
c_{33} &= \#\{k \mid d_{3k} < d_{3k}\} + \#\{k \mid d_{k3} > d_{k3}\} = 0
\end{aligned}$$

Hence, the cost matrix $\boldsymbol{C}$ in this case is the following.

$$\boldsymbol{C} = \begin{pmatrix} 0 & 4 & 3 \\ 0 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix}$$

□

We will return to this example later, but first we need to introduce some more notation and also a result showing that the matrix $\boldsymbol{C}$ can in fact be used to calculate the number of violations in $\boldsymbol{D}$ from Hillside form. Following the lines of Pedings et. al., we introduce yet another $n \times n$-matrix, $\boldsymbol{X}$, and name it the *decision matrix*. Each entry in $\boldsymbol{X}$, x_{ij}, will be a decision variable, meaning that x_{ij} is either 0 or 1 for $1 \leq i, j \leq n$. Since what we really want to decide is the ranking of the n items, we define these variables as

$$x_{ij} = \begin{cases} 1 & \text{if item } i \text{ is ranked above item } j \\ 0 & \text{otherwise} \end{cases}$$

In the $n = 3$ case, we have 3 items we would like to rank. And if the items 1, 2 and 3 were ranked in that order, the matrix $\boldsymbol{X}$ would look like this:

$$\boldsymbol{X} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{4.1}$$

More generally, we could have items $\{1, 2, \ldots, n\}$ ranked in that order (i.e. that item 1 is ranked as 1, item 2 as 2 and so on). Then the $n \times n$-matrix $\boldsymbol{X}$ is a strictly upper triangular matrix with 1s above the diagonal:

$$\boldsymbol{X} = \begin{pmatrix} 0 & 1 & 1 & \ldots & 1 \\ 0 & 0 & 1 & \ldots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ 0 & 0 & 0 & \ldots & 0 \end{pmatrix}$$

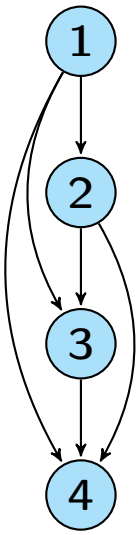
Since the ranking method in question here is a *minimum violations ranking method* (MVR method), we would like to minimize the number of violations from hillside form for a given matrix $\boldsymbol{D}$. Using our newly defined matrices $\boldsymbol{C}$ and $\boldsymbol{X}$ we shall see that we can write this problem as a binary integer linear program (BILP).

Before we can state the BILP formulation however, we have to introduce some additional constraints on the decision variables x_{ij} in $\boldsymbol{X}$. Recall that $x_{ij} = 1$ if item i is ranked above item j, 0 if this is not the case. In other words, the variables x_{ij} must be binary. Now we shall see that these variables must respect two other constraints as well (to be used for our purpose), namely *antisymmetry* and *transitivity*.

That x_{ij} must be antisymmetric is quite straightforward to see. Given two items, item i and item j, one of them must be ranked above the other, thus we

must have for all distinct pairs (i, j) that $x_{ij} + x_{ji} = 1$ (as one the items must be the 'best' of the two).

The transitivity might be a little harder to convince oneself of. The constraint we now want to add is this: $x_{ij} + x_{jk} + x_{ki} \leq 2$ for all distinct triples (i, j, k). To see why this must be true for all distinct triples (i, j, k), observe that for it to be false, we must have that $x_{ij} = x_{jk} = x_{ki} = 1$. This would mean that item i is ranked above j, item j is ranked above item k and k is ranked above item i. Hence item i is ranked above itself, which is impossible.



These arguments may be visualized as *dominance graphs*, to the right we see a dominance graph with four nodes. If a dominance graph contains an upwards edge, then there is a triple (i, j, k) that violates the equation.

Using these constraints for $\boldsymbol{X}$ we can write our BILP as the following:

Problem 4.10.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$x_{ij} + x_{ji} = 1$ for all distinct pairs (i, j) (antisymmetry)

$x_{ij} + x_{jk} + x_{ki} \leq 2$ for all distinct triples (i, j, k) (transitivity)

$x_{ij} \in \{0, 1\}$ (binary)

This BILP will aim to find a matrix $\boldsymbol{X}$ that is a reordering of an upper-triangular matrix with only 1s in the upper triangle (we saw the 3×3 case of this in equation 4.1). Thus the BILP will produce a unique ranking (since the row and column sums will be unique) of the n items, which we can read from $\boldsymbol{X}$.

Now we state a theorem that shows why we have created the matrices $\boldsymbol{C}$ and $\boldsymbol{X}$ as we have. But first we must make it clear what we mean by a hillside violation. We denote the number of hillside violations in $\boldsymbol{D}$ by $\#\text{viol}(\boldsymbol{D})$.

Lemma 4.11. We have the following equality:

$$\#\text{viol}(\boldsymbol{D}) = \#\{k \mid d_{ik} < d_{jk} \quad \forall i < j\} + \#\{d_{ki} > d_{kj} \quad \forall i < j\}$$

Proof. What is a violation from hillside form? We turn to the definition of hillside form and realise that a violation occurs in the columns of $\boldsymbol{D}$ whenever an entry a is smaller than an entry b, where a is above b in the same column. Likewise we have violations in the rows when an entry c which is to the left of an entry d, is larger than the said number d. Instead of writing a, b, c, d for the entries, we may use d_{ij}. So violations in the columns are ks such that $d_{ik} < d_{jk}$ for $i < j$. Similarly, violations in rows are ks such that $d_{ki} > d_{kj}$ for $i < j$. Put together this gives

$$\#\text{viol}(\boldsymbol{D}) = \#\{k \mid d_{ik} < d_{jk} \quad \forall i < j\} + \#\{d_{ki} > d_{kj} \quad \forall i < j\}$$

□

Theorem 4.12. Let $\boldsymbol{D}$ be an $n \times n$ point differential matrix. Let $\boldsymbol{C}$ be the cost matrix defined in Definition 4.8. Then $\boldsymbol{C}$ can be used to compute the number of violations from hillside form for $\boldsymbol{D}$.

Proof. Assume $\boldsymbol{X}$ is the matrix associated with the ranking of the n elements (the matrix of decision variables as before). Further we assume that the ranking of the n elements is $1, 2, \ldots, n$. Then $\boldsymbol{X}$ will be upper triangular with only 1s in the upper triangle, as we saw for the 3×3 case in 4.1. Let $\boldsymbol{C}$ be the cost matrix for a given point differential matrix $\boldsymbol{D}$. If we now use $\boldsymbol{C}$ and $\boldsymbol{X}$ (as defined) in the BILP (Problem (4.10)) we get that the objective function (the function we want to minimize) is

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} = (c_{11} + c_{12} + \cdots + c_{1n}) + (c_{23} + \cdots + c_{2n}) + \cdots + c_{n-1,n}$$

which we can see is the sum of all the elements in the strict upper triangluar part of $\boldsymbol{C}$. Now we use the definition of $\boldsymbol{C}$ (i.e. definition of each element c_{ij}) and get

$$\begin{aligned}
&= ((\#\{k \mid d_{1k} < d_{1k}\} + \#\{k \mid d_{k1} > d_{k1}\}) + (\#\{k \mid d_{1k} < d_{2k}\} + \#\{k \mid d_{k1} > d_{k2}\}) \\
&\quad + \cdots + (\#\{k \mid d_{1k} < d_{nk}\} + \#\{k \mid d_{k1} > d_{kn}\})) + \cdots + \ldots \\
&\quad \ldots (\#\{k \mid d_{(n-1),k} < d_{nk}\} + \#\{k \mid d_{k,(n-1)} > d_{kn}\}) \\
&= \#\{k \mid d_{ik} < d_{jk} \quad \forall i < j\} + \#\{d_{ki} > d_{kj} \quad \forall i < j\} \\
&= \#\text{viol}(\boldsymbol{D})
\end{aligned}$$

Where the last equality comes from Lemma 4.11.

Originally we assumed that the ranking was $\{1, 2, \ldots, n\}$, but we can do this without loss of generality as we may reorder the matrices $\boldsymbol{D}, \boldsymbol{C}$ and $\boldsymbol{X}$ so the associated ranking indeed is $\{1, 2, \ldots, n\}$. Thus we have showed that the $\boldsymbol{C}$ matrix can be used to calculate $\#\text{viol}(\boldsymbol{D})$. □

Now that we know that $\boldsymbol{C}$ may be used to calculate the number of violations from hillside form we can turn again to our BILP where we would like to minimize $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$. From Theorem 4.12 we then get the following:

Corollary 4.13. The solution of Problem 4.10 gives a ranking that minimizes the number of violations from hillside form.

In Appendix A.1 we give programs that convert between a decision matrix $\boldsymbol{X}$ and the corresponding ranking $\boldsymbol{\pi}$, in addition to a program that calculates the minimum number of violations from hillside form for a matrix according to Theorem 4.12.

4.3 Implementation of MVR method in OPL

The main reason for reformulating the MVR from a QIP to a BILP was the computational advantages, and the fact that it is easier to implement BILPs than QIPs. In this section we will try to implement the MVR method of Pedings et. al. in the optimization software CPLEX using the modeling language OPL, exploiting the BILP formulation of the MVR problem we found in the previous section. Since no actual code is given in [PLY12], we create our own programs here.

In Problem 4.10 we take as given the cost matrix $\boldsymbol{C}$, so to implement the BILP in OPL we must first have the cost matrix. One possible way of finding a cost matrix for a point differential matrix can be found in Appendix A.1.1. Below we have the implementation of Problem 4.10 in OPL to use in CPLEX. Notice that this is only the `mod`-file. Whenever one would like to run this program one has to put the specifics of the current problem in a separat file (a `dat`-file) and make a configuration with the `mod`- and `dat`-file. This is just how CPLEX works. As we see, the `mod`-file is quite similar to the mathematical formulation of the minimization problem.

Listing 4.1: `bilp.mod`

```
1 int n = ...;
2 range N = 0..n-1;
```

```
3
4  int c[N][N] = ...;        /* cost matrix */
5
6  dvar boolean x[N][N];     /* boolean will ensure that all values are 0 or 1*/
7
8  minimize
9  sum(i in N, j in N) c[i][j]*x[i][j];
10
11 subject to {
12
13  forall(i in N)
14    forall(j in N)
15      if (i != j) x[i][j] + x[j][i] == 1;
16
17  forall(i in N)
18    forall(j in N)
19      forall(k in N)
20      if (i != j && j != k && i != k) x[i][j] + x[j][k] + x[k][i] <= 2;
21 }
```

This implementation takes a cost matrix $\boldsymbol{C}$ and returns the decision matrix $\boldsymbol{X}$ as discussed in the last section. The $\boldsymbol{X}$ matrix will be a reordering of a strictly upper triangular matrix with only 1s above the diagonal. This produces our ranking. The team with the lowest column sum of $\boldsymbol{X}$ is ranked as number 1, the team with next lowest column sum is ranked second and so on. Now we will check how the program performs on some examples.

Example 4.14. Recall the situation in Example 4.9. The cost matrix $\boldsymbol{C}$ was calculated, and this matrix is our input to the BILP-program above. To illustrate the usage of CPLEX we show here what the `dat`-file is for this particular example.

```
1  n = 3;
2
3  c = [[0 4 3]
4       [0 0 0]
5       [0 3 0]
6       ];
```

To solve this optimization problem one runs a configuration with the `dat`-file above and the `mod`-file `bilp.mod`. The result is the following matrix.

$$\boldsymbol{X} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

For this small example it is straight-forward to see what the column sums are and find the ranking from them. For larger examples a few lines of code will solve this (Appendix A.1.2). The column sums are 2, 0 and 1, respectively. This means that our ranking is $\boldsymbol{r}^T = \begin{pmatrix} 2 & 3 & 1 \end{pmatrix}$, i.e. team 2 is ranked as number 1, team 3 as number 2 and team 1 as number 3. If we go back to Example 4.9 we see that this ranking seems to reflect the actual situation nicely. □

Example 4.15. In the article [PLY12] the authors have an example from the 2008-2009 Southern Conference (SoCon) basketball season. They do not report the point differential matrix, but rather the cost matrix $\boldsymbol{C}$ it produces.

$$\boldsymbol{C} = \begin{pmatrix} 0 & 15 & 15 & 14 & 17 & 7 & 4 & 4 & 9 & 2 & 10 & 11 \\ 8 & 0 & 10 & 12 & 18 & 6 & 3 & 3 & 11 & 3 & 7 & 8 \\ 5 & 11 & 0 & 9 & 14 & 6 & 2 & 4 & 9 & 2 & 5 & 9 \\ 5 & 9 & 9 & 0 & 15 & 5 & 0 & 2 & 6 & 3 & 6 & 5 \\ 2 & 2 & 5 & 3 & 0 & 2 & 1 & 2 & 0 & 1 & 1 & 2 \\ 10 & 14 & 16 & 17 & 18 & 0 & 7 & 7 & 12 & 4 & 13 & 15 \\ 15 & 18 & 18 & 20 & 20 & 13 & 0 & 8 & 16 & 10 & 15 & 15 \\ 15 & 20 & 18 & 18 & 20 & 13 & 10 & 0 & 15 & 11 & 14 & 18 \\ 10 & 9 & 11 & 14 & 19 & 7 & 4 & 7 & 0 & 2 & 10 & 9 \\ 17 & 17 & 18 & 18 & 20 & 16 & 7 & 9 & 15 & 0 & 13 & 14 \\ 10 & 14 & 14 & 10 & 18 & 8 & 4 & 4 & 12 & 7 & 0 & 12 \\ 10 & 12 & 11 & 12 & 17 & 7 & 4 & 4 & 10 & 6 & 8 & 0 \end{pmatrix}$$

Pedings et. al. furthermore writes that solving the BILP with this cost matrix produces an objective value of 351, along with a matrix $\boldsymbol{X}$ and the corresponding ranking. Since we now have written a program that solves the BILP instance of the MVR problem, we hope that the program will produce the same solutions as Pedings et. al. reports.

We feed the given cost matrix to the implementation in CPLEX, and then uses MATLAB to find the corresponding ranking. Our results are these:

$$\boldsymbol{X} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

The corresponding ranking we get is this

$$\boldsymbol{r}^T = \begin{pmatrix} 5 & 4 & 3 & 9 & 2 & 12 & 11 & 1 & 6 & 10 & 7 & 8 \end{pmatrix}$$

This ranking is just a list of the column sums, starting with the column with lowest column sum. This means that since column (hence team) 5 has column sum equal to zero, team 5 is ranked first. Team 4 has column sum equal to 1, and hence this team is placed second in the ranking. And so on. In the next chapter we will introduce another notation for a ranking.

The big question now is of course: is what we got in Example 4.15 the same result as Pedings et. al. got in their article? With one exception the results are equal. Our CPLEX implementation also reports that the optimal objective value is 351, which is pleasing. However, the matrix $\boldsymbol{X}$ above is slightly different from the one in the article. The $(1, 11)$-entry and the $(11, 1)$-entry has switched places (i.e. a 1 is a 0 and vice versa).

This results in team 11 being ranked before team 1, but in the article this is the other way around. The only difference in terms of the ranking is thus that team 11 and team 1 has switched places on the ranking list. But since both solutions have the (same) optimal objective value, we are satisfied with our implementation of the BILP formulation. □

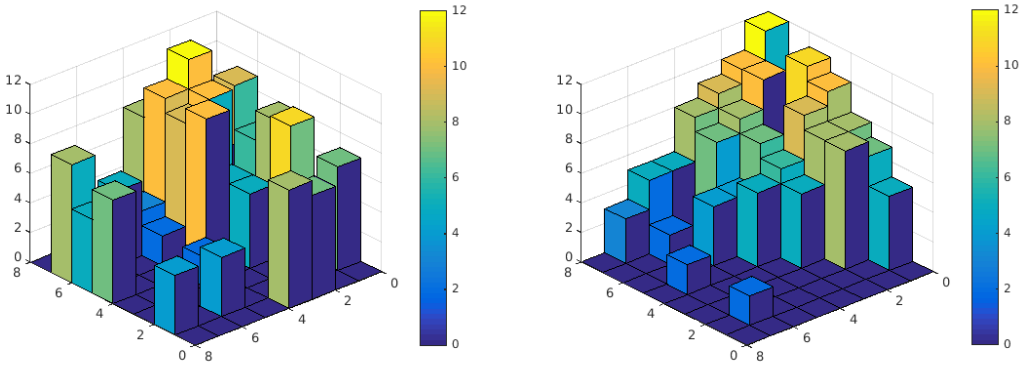
Example 4.16. Assume we have the following point differential $\boldsymbol{D}$ and the corresponding cost matrix $\boldsymbol{C}$.

$$\boldsymbol{D} = \begin{pmatrix} 0 & 7 & 0 & 0 & 8 & 6 & 9 & 5 \\ 0 & 0 & 0 & 0 & 2 & 0 & 5 & 0 \\ 7 & 11 & 0 & 5 & 5 & 10 & 12 & 8 \\ 8 & 0 & 0 & 0 & 10 & 9 & 10 & 8 \\ 0 & 0 & 0 & 2 & 0 & 2 & 3 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 7 & 5 & 8 & 0 \end{pmatrix} \qquad \boldsymbol{C} = \begin{pmatrix} 0 & 1 & 8 & 7 & 2 & 0 & 0 & 0 \\ 9 & 0 & 11 & 9 & 4 & 4 & 0 & 8 \\ 1 & 0 & 0 & 2 & 0 & 0 & 0 & 1 \\ 2 & 1 & 6 & 0 & 2 & 1 & 0 & 2 \\ 9 & 6 & 12 & 9 & 0 & 6 & 0 & 8 \\ 10 & 4 & 12 & 9 & 4 & 0 & 0 & 8 \\ 12 & 9 & 14 & 12 & 9 & 9 & 0 & 11 \\ 7 & 1 & 9 & 8 & 2 & 0 & 0 & 0 \end{pmatrix}$$

We run the program in CPLEX and get that the reordering that minimizes the number of violations from hillside form for $\boldsymbol{D}$ is $\begin{pmatrix} 3 & 6 & 1 & 2 & 7 & 5 & 8 & 4 \end{pmatrix}$. We symmetrically reorder $\boldsymbol{D}$ to $\boldsymbol{D'}$ according to that vector. $\boldsymbol{D'}$ has 30 violations.

$$\boldsymbol{D'} = \begin{pmatrix} 0 & 5 & 7 & 8 & 10 & 11 & 5 & 12 \\ 0 & 0 & 8 & 8 & 9 & 0 & 10 & 10 \\ 0 & 0 & 0 & 5 & 6 & 7 & 8 & 9 \\ 0 & 0 & 0 & 0 & 5 & 4 & 7 & 8 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 5 \\ 0 & 2 & 0 & 0 & 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

To show the differences between $\boldsymbol{D}$ and $\boldsymbol{D}'$ we again turn to MATLABs city-plot.



□

We will return to our CPLEX-program when we in Chapter 5 develop a slightly different ranking method than the one in [PLY12] and when we in Section 5.6 compare these two ranking methods. This new ranking method will also exploit the concept of hillside form, but in a different way than that of Pedings et. al.

Chapter 5

A minimum hillside distance ranking method

In Chapter 4 we looked at the article [PLY12] and the ranking method it describes. The central part of the method was the notion of hillside form for quadratic matrices. In this chapter we will start with this notion and see if we can solve a different mathematical problem to find a ranking. In order to this, we define a notion of distance from hillside form, which to our knowledge is original.

Along the way we will make use of some definitions and results from abstract algebra (permutations, cycles, etc.), our primary source on this will be the book *A first course in abstract algebra* by John B. Fraleigh ([Fra67]).

5.1 Distance from hillside form

In their article, Pedings, Langville and Yamamoto ([PLY12]) introduce the notion of hillside form. We repeat their definition here for convenience.

Definition 5.1. An $n \times n$-matrix $\boldsymbol{D} = [d_{ij}]_{i,j=1}^{n}$ is in *hillside form* if

$$\begin{aligned} d_{ij} &\leq d_{ik} \quad \forall i \text{ and } \forall j \leq k \quad \text{(ascending order across rows)} \\ d_{ij} &\geq d_{kj} \quad \forall j \text{ and } \forall i \leq k \quad \text{(descending order down columns)} \end{aligned}$$

Through the MVR method they construct, Pedings et. al. use the concept of number of violations from hillside form for a matrix. Now, rather than use the number of violations from hillside form, we would like to speak of *distance from hillside form* for a matrix. Furthermore, we would like to create a ranking method from the one in [PLY12], also starting with a point differential matrix $\boldsymbol{D}$

and exploiting the distance from hillside form to reach a ranking of the elements which pairwise comparisons $\boldsymbol{D}$ is made up of. This new method of ours will solve a different mathematical problem than the one in [PLY12], even though the ideal solution we seek is the same as in [PLY12].

To motivate the definition of distance from hillside form, we consider the following example. The matrices considered have the same number of violations, but we see that they reflect two rather different situations. In one we barely get an upset, but in the other we have a crushing upset. If we had a measure on how far rom hillside form a matrix was which took into account the "size" of the violations, we could have separated these two situations.

Example 5.2.

$$\boldsymbol{A} = \begin{pmatrix} 0 & 3 & 0 & 9 \\ 0 & 0 & 4 & 7 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} 0 & 3 & 0 & 9 \\ 0 & 0 & 4 & 7 \\ 17 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Both matrices are 7 violations from hillside form, according to Definition 5.1. □

Since we would like the notion of distance from hillside form to be somewhat intuitive, we start by establishing some properties that we would like this distance to have. Firstly, we want the distance to be a function that takes an $n \times n$-matrix[1] as an argument and returns a value we call the distance from hillside form for the matrix $\boldsymbol{D}$.

Naturally, we would like this distance function to return 0 if the matrix in question indeed is in hillside form. In addition we want the function to return a positive number if the matrix is not in hillside form, and also that this number is a (relatively speaking) low number for a matrix that is close to hillside form, and a large number when the matrix is far from hillside form.

Based on these properties we make the following definiton. We denote by $\mathbb{M}_n(\mathbb{R})$ the space of $n \times n$ matrices with all real entries.

Definition 5.3. Let $\boldsymbol{D} = [d_{ij}]_{i,j=1}^n$ be an $n \times n$-matrix. Let $h : \mathbb{M}_n(\mathbb{R}) \to \mathbb{R}$ be defined by

$$h(\boldsymbol{D}) = \sum_{i=1}^{n} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \max\{d_{ij} - d_{ik}, 0\} + \max\{d_{ki} - d_{ji}, 0\}$$

[1]That the input matrix needs to be square follows from the fact that the point differential matrix $\boldsymbol{D}$ is created from pairwise comparisons of n items.

We call $h(\boldsymbol{D})$ the *distance of* $\boldsymbol{D}$ *from hillside form.*

In the definition above one can think of $\max\{d_{ij} - d_{ik}, 0\}$ as the contribution from the rows. It is 0 if $d_{ij} \leq d_{ik}$, which they are if $\boldsymbol{D}$ is in hillside form, and we get a positive contribution when an element is larger than some element to its right. Similar arguments apply to the column contribution, $\max\{d_{ki} - d_{ji}, 0\}$, except here we get a positive contribution only if an element below another element is the largest of two.

One can think of the distance from hillside form for a matrix as the sum of a measure on how wrong each entry in the matrix is. A wrong entry in a matrix means either an upset or a weak win, which occurs whenever a team performs worse or better than what is expected of it and the wrongness depends on how much they over-/underperformed. And what is expected of a team is given by that team's ranking. When no unexpected results occur our matrix is in hillside form, as the following proposition shows.

Proposition 5.4. $h(\boldsymbol{D}) = 0$ if and only if $\boldsymbol{D}$ is in hillside form.

Proof. Assume $\boldsymbol{D}$ is an $n \times n$-matrix. Then we have the following:

$$
\begin{aligned}
& h(\boldsymbol{D}) = 0 \\
& \qquad \Updownarrow \qquad \textit{(by Definition 5.3)} \\
& \sum_{i=1}^{n} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \max\{d_{ij} - d_{ik}, 0\} + \max\{d_{ki} - d_{ji}, 0\} = 0 \\
& \qquad \Updownarrow \qquad \textit{(for any } a \in \mathbb{R}\textit{, } \max\{a, 0\} \geq 0\textit{)} \\
& \max\{d_{ij} - d_{ik}, 0\} = 0 \text{ and } \max\{d_{ki} - d_{ji}, 0\} = 0, \quad i \in [1, n], j \in [1, n-1], k \in [j+1, n] \\
& \qquad \Updownarrow \\
& d_{ij} - d_{ik} \leq 0 \text{ and } d_{ki} - d_{ji} \leq 0, \quad i \in [1, n], j \in [1, n-1], k \in [j+1, n] \\
& \qquad \Updownarrow \\
& d_{ij} \leq d_{ik} \text{ and } d_{ji} \geq d_{ki}, \quad i \in [1, n], j \in [1, n-1], k \in [j+1, n]
\end{aligned}
$$

The last line is equivalent with the hillside definition, as we can check. We have that $d_{ij} \leq d_{ik}$ and since we also have that $1 \leq i \leq n$, $1 \leq j \leq n-1$ and $j+1 \leq k \leq n$, we have ascending order across rows. Likewise, we have that $d_{ji} \geq d_{ki}$ with the same conditions for i, j and k. This means that we must have descending order down columns. A matrix that has ascending order across rows and descending order down columns is, by definition, a matrix in hillside form. □

Example 5.5. Assume we have the 3×3-matrix given by

$$\boldsymbol{D} = \begin{pmatrix} 0 & 4 & 7 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{pmatrix}$$

We see that $\boldsymbol{D}$ is in hillside form. Nonetheless we check that our function h returns 0.

$$\begin{aligned} h(\boldsymbol{D}) &= \sum_{i=1}^{3} \sum_{j=1}^{2} \sum_{k=j+1}^{3} \max\{d_{ij} - d_{ik}, 0\} + \max\{d_{ki} - d_{ji}, 0\} \\ &= \max\{0-4,0\} + \max\{0-0,0\} + \max\{0-7,0\} \\ &\quad + \max\{0-0,0\} + \max\{4-7,0\} + \max\{0-0,0\} \\ &\quad + \max\{0-0,0\} + \max\{0-4,0\} + \max\{0-3,0\} \\ &\quad + \max\{0-4,0\} + \max\{0-3,0\} + \max\{0-0,0\} \\ &\quad + \max\{0-0,0\} + \max\{0-0,0\} + \max\{0-0,0\} \\ &\quad + \max\{0-0,0\} + \max\{0-0,0\} + \max\{0-0,0\} \\ &= 0 \end{aligned}$$

The distance from hillside form is 0, thus $\boldsymbol{D}$ is in hillside form by Proposition 5.4. □

Example 5.6. Assume we have the 3×3-matrix given by

$$\boldsymbol{D} = \begin{pmatrix} 0 & 4 & 7 \\ 2 & 0 & 3 \\ 0 & 0 & 0 \end{pmatrix}$$

We see easily that $\boldsymbol{D}$ is not in hillside form. But how far from hillside form, in our terms, is $\boldsymbol{D}$? Let us apply Definition 5.3.

$$h(\boldsymbol{D}) = \sum_{i=1}^{3} \sum_{j=1}^{2} \sum_{k=j+1}^{3} \max\{d_{ij} - d_{ik}, 0\} + \max\{d_{ki} - d_{ji}, 0\} = 4$$

This might not come as a surprise, as we see that 2 is the only element misplaced for $\boldsymbol{D}$ to be in hillside form. The 2 makes two contributions, once in the row it is located and once in the column it is located. □

Example 5.7. We consider again the matrices in Example 5.2. They have the same number of violations, but they are not equally close to hillside form when we apply our new distance notion. $\boldsymbol{A}$ has distance 16 from hillside form, and $\boldsymbol{B}$ has distance 91 from hillside form. This means that $\boldsymbol{A}$ is closer to hillside form than $\boldsymbol{B}$ is. □

The primary reason for us wanting to have our matrix in hillside form is because one can easily read a ranking of a matrix in hillside form. The item in row 1 will be ranked as number 1, the item in the second row will be second and so on. If we have a matrix that is not in hillside form, we would like to take it closer to hillside form (if possible) by permuting the matrix (i.e. permuting the rows and columns). We therefore start by summarizing some of the theory about permutations.

5.2 Permutations and cycles

Definition 5.8. A *permutation* of a set A is a function $\boldsymbol{\pi} : A \to A$ that is both one-to-one and onto. (If the set A is finite it is sufficient that $\boldsymbol{\pi}$ is either one-to-one or onto for $\boldsymbol{\pi}$ to be a permutation (as the other property will follow from the first in this case)). When $A = \{1, 2, \ldots, n\}$ is finite we denote by S_n the set of all permutations of A.

More loosely speaking we say that a permutation of a set A is a *reordering* of A. Let us now say that $A = \{1, 2, \ldots, n\}$ and that these numbers represent n different items. Then we can think of a permutation, $\boldsymbol{\pi}$, of those n items by

$$\boldsymbol{\pi} = \begin{pmatrix} 1 & 2 & \ldots & n \\ \boldsymbol{\pi}(1) & \boldsymbol{\pi}(2) & \ldots & \boldsymbol{\pi}(n) \end{pmatrix}$$

This means that item 1 is permuted to $\boldsymbol{\pi}(1)$, item 2 is permuted to $\boldsymbol{\pi}(2)$ and so on. We can shorten this notation by simply writing

$$\boldsymbol{\pi} = \begin{pmatrix} \boldsymbol{\pi}(1) & \boldsymbol{\pi}(2) & \ldots & \boldsymbol{\pi}(n) \end{pmatrix}$$

Since we want to apply this theory to ranking, we make the following definition.

Definition 5.9. If A is a set with n elements, and $\boldsymbol{\pi} : A \to A$ is a permutation of A, then

$$\boldsymbol{\pi}(i) = \text{ ranking of element i}, \qquad 1 \leq i \leq n$$

In the previous chapter we used a different notation for ranking. There we used $\boldsymbol{r}$ and interpreted $\boldsymbol{r}(i)$ as team $\boldsymbol{r}(i)$ being in ith place. But with this new permutation notation for ranking, we interpret $\boldsymbol{\pi}(i)$ as team i being in $\boldsymbol{\pi}(i)$th place. We will be consistent with this notation throughout this thesis, so hopefully no misconceptions will arise. Nevertheless we have the following example to illustrate the differences between these notations. In Appendix A.2.2 a program for converting from $\boldsymbol{r}$-notation to $\boldsymbol{\pi}$-notation is given.

Example 5.10. Assume we have the following rankings

$$\boldsymbol{r}^T = \begin{pmatrix} 3 & 7 & 2 & 1 & 6 & 4 & 5 \end{pmatrix}$$
$$\boldsymbol{\pi} = \begin{pmatrix} 3 & 7 & 2 & 1 & 6 & 4 & 5 \end{pmatrix}$$

These rankings are different in our eyes. In the $\boldsymbol{r}$-case we have that team 3 is placed first, team 7 is placed second and so on, until team 5 is placed in 7th place. In the $\boldsymbol{\pi}$-case we have a different ranking. Here $\boldsymbol{\pi}(1) = 3$, so team 1 is in third place. Since $\boldsymbol{\pi}(2) = 7$, team 2 is in 7th place. And so on, until team 7 is placed fifth as $\boldsymbol{\pi}(7) = 5$. □

A permutation $\boldsymbol{\pi}$ gives rise to a corresponding permutation matrix $\boldsymbol{P_\pi}$. We let $\boldsymbol{e_j}$ be the row vector with all entries equal to zero, except at entry j where it is 1. Now we make the following definition.

Definition 5.11. If $\boldsymbol{\pi}$ is a permutation of n elements, then its *permutation matrix* is given by

$$\boldsymbol{P_\pi} = \begin{pmatrix} \boldsymbol{e_{\pi(1)}} \\ \boldsymbol{e_{\pi(2)}} \\ \vdots \\ \boldsymbol{e_{\pi(n)}} \end{pmatrix}$$

Example 5.12. If we were given the permutation $\boldsymbol{\pi} = \begin{pmatrix} 3 & 2 & 4 & 1 \end{pmatrix}$, the corresponding permutation matrix, $\boldsymbol{P_\pi}$, will be given as

$$\boldsymbol{P_\pi} = \begin{pmatrix} \boldsymbol{e_{\pi(1)}} \\ \boldsymbol{e_{\pi(2)}} \\ \boldsymbol{e_{\pi(3)}} \\ \boldsymbol{e_{\pi(4)}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{e_3} \\ \boldsymbol{e_2} \\ \boldsymbol{e_4} \\ \boldsymbol{e_1} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

□

It is not hard to see that a permutation matrix is a doubly stochastic matrix and that it can be obtained by interchanging rows of an identity matrix. We contemplate now on what happens to a matrix when we multiply it by a permutation matrix. If one premultiplies a matrix $\boldsymbol{D}$ by a permutation matrix $\boldsymbol{P_\pi}$, $\boldsymbol{P_\pi D}$, one gets a permutation of the rows in $\boldsymbol{D}$ according to $\boldsymbol{\pi}$. Likewise, if one postmultiplies $\boldsymbol{D}$ by $\boldsymbol{P_\pi}$, $\boldsymbol{DP_\pi}$, the columns of $\boldsymbol{D}$ are permuted.

If one both pre- and postmultiplies a matrix by the same permutation matrix, we do what is known as a *symmetric permutation* or a *symmetric reordering*. What this means is that we permute both the rows and the columns by the same

permutation $\boldsymbol{\pi}$. In symbols we get that ${\boldsymbol{P}_{\boldsymbol{\pi}}}^T \boldsymbol{D} \boldsymbol{P}_{\boldsymbol{\pi}}$.

Now we introduce some more notions from algebra that will be of use to us. Later we will talk about k-cycles (mostly for $k = 2, 3$), and to talk about cycles we must first introduce the concept of orbits for a permutation.

Given a permutation $\boldsymbol{\pi}$ of a set A, what might be a natural partition of A in terms of $\boldsymbol{\pi}$? In a sense we would like two elements of A, call them a and b, to belong to the same component of A if one of them, say b, can be "reached" from the other one, a, using the permutation $\boldsymbol{\pi}$. Formally, we would like a and b to be in the same "permutation component" of A if $b = \boldsymbol{\pi}^n(a)$ for some $n \in \mathbb{Z}$. This criteria for partitioning A leads to an equivalence relation on the elements of A.

Proposition 5.13. Let $\boldsymbol{\pi}$ be a permutation of a set A. Then for $a, b \in A$, let $a \sim b$ if and only if $b = \boldsymbol{\pi}^n(a)$ for some $n \in \mathbb{Z}$. The relation $\sim$ is an equivalence relation.

Proof. An equivalence relation must satisfy the three properties reflexivity, symmetry and transitivity. We check that these three properties hold for $\sim$. Assume that $a, b, c \in A$ and that $\boldsymbol{\pi}$ is a permutation of A.

> *Reflexive: Is $a \sim a$?* Since $a = \boldsymbol{\pi}^0(a)$ and $0 \in \mathbb{Z}$, we have that $a \sim a$.
>
> *Symmetric: If $a \sim b$, is also $b \sim a$?* Assume $a \sim b$. By definition of $\sim$ this means that $b = \boldsymbol{\pi}^n(a)$ for some $n \in \mathbb{Z}$. But we can take the inverse permutation and get that $a = \boldsymbol{\pi}^{-n}(b)$. Since $-n \in \mathbb{Z}$, we must have that $b \sim a$.
>
> *Transitive: If $a \sim b$ and $b \sim c$, is then also $a \sim c$?* Suppose $a \sim b$ and $b \sim c$. Then, by definition, we have that $b = \boldsymbol{\pi}^n(a)$ and $c = \boldsymbol{\pi}^m(b)$ for some $m, n \in \mathbb{Z}$. Putting this together we get that $c = \boldsymbol{\pi}^m(b) = \boldsymbol{\pi}^m(\boldsymbol{\pi}^n(a)) = \boldsymbol{\pi}^{m+n}(a)$. Since $m + n \in \mathbb{Z}$, we have that $a \sim c$.

□

Definition 5.14. Let $\boldsymbol{\pi}$ be a permutation of a set A. The equivalence classes in A determined by the equivalence relation in Proposition 5.13 are called the *orbits* of $\boldsymbol{\pi}$.

Definition 5.15. A permutation $\boldsymbol{\pi}$ in S_n is a *cycle* if it has at most one orbit containing more than one element. The length of a cycle is the number of elements in its largest orbit.

Note that instead of 2-cycle (or cycle of length 2) we usually say *transposition*. Recall that we denote by S_n the set of all permutations of a finite set $A = \{1, 2, \ldots, n\}$. Now we are going to show some properties of permutations relating to cycles.

Proposition 5.16. Any permutation in S_n can be written as a product of disjoint cycles.

Proof. Assume $\boldsymbol{\pi}$ is a permutation in S_n. Furthermore let $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_r$ be the orbits of $\boldsymbol{\pi}$. Now, define μ_i as the following cycle:

$$\mu_i(x) = \begin{cases} \boldsymbol{\pi}(x) & \text{if } x \in \mathcal{O}_i \\ x & \text{otherwise} \end{cases}$$

By the construction of μ_i we get that $\boldsymbol{\pi} = \mu_1\mu_2 \ldots \mu_r$. Since the orbits $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_r$ arise from distinct equivalence classes, they are disjoint, hence the cycles $\mu_1, \mu_2, \ldots, \mu_r$ are disjoint as well. □

Corollary 5.17. Any permutation in S_n, $n \geq 2$, can be written as a product of transpositions.

Proof. By the previous proposition it suffices to show that this is true for cycles. This can be done by swapping all pairs of adjacent elements. □

This can also readily be seen when we think of a permutation of n elements as a rearrangement of n elements. Any rearrangement of the n elements can be obtained by successively swapping two and two elements.

5.3 Creating a ranking method

How are we going to apply our knowledge of permutations to the hillside form? Well, for a start we could try to find a permutation $\boldsymbol{\pi}$ such that $\boldsymbol{P_\pi}^T \boldsymbol{D} \boldsymbol{P_\pi}$ is closer to hillside form than $\boldsymbol{D}$ is. More precisely, we would like to find a permutation $\boldsymbol{\pi}$ such that $h(\boldsymbol{D}) > h(\boldsymbol{D_\pi})$, where $\boldsymbol{D_\pi} = \boldsymbol{P_\pi}^T \boldsymbol{D} \boldsymbol{P_\pi}$. In general we have the following problem that we would like to solve.

Problem 5.18.

$$\underset{\boldsymbol{\pi} \in S_n}{\text{minimize}} \quad h(\boldsymbol{D_\pi})$$

Let us say we have found a $\boldsymbol{\pi}$ such that $h(\boldsymbol{D}_{\boldsymbol{\pi}}) < h(\boldsymbol{D})$. Then we can proceed to find an even permutation $\boldsymbol{\pi}'$ such that $h(\boldsymbol{D}_{\boldsymbol{\pi}'}) < h(\boldsymbol{D}_{\boldsymbol{\pi}})$. We may continue in this fashion until we no longer can find a permutation that can improve our distance from hillside form (one possibility being that $h(\boldsymbol{D}_{\boldsymbol{\pi}}) = 0$ for some permutation $\boldsymbol{\pi}$, which means that we have gotten $\boldsymbol{D}_{\boldsymbol{\pi}}$ to hillside form and life is good).

This kind of procedure, where one in each step only see one step into the future and tries to optimize with respect to only the next step, is an example of a so called *local method*.

The procedure described above may be implemented in some programming language. Pseudocode for such a program can be found in Algorithm 1. This algorithm will be the core of the next subsections, as we will start with a quite naive program and try to improve it along the way.

It is mainly two parts of Algorithm 1 that we will concentrate on. In line 4 in the program we need to choose some initial permutation (ranking) $\boldsymbol{\pi}$, and at the iteration in line 6 we need to choose a new 'nearby' permutation, $\boldsymbol{\pi}'$. There are many ways in which we may search for or choose these permutations, and over the next subsections we will take a look at some of these possibilities.

Algorithm 1 Minimizing distance to hillside form (solving Problem (5.18))

```
1: if h(D) == 0 then
2:    break                          % D is in hillside form
3: else
4:    find some initial ranking/permutation π
5:    while h(D) ≠ 0 do
6:       for π' 'nearby' π do
7:          D' = P_{π'}^T D P_{π'}
8:          if h(D') < h(D) then
9:             π = π'
10:            D = D'
11:         end if
12:      end for
13:      if no improving 'nearby' π' was found then
14:         break
15:      end if
16:   end while
17: end if
```

We will start by letting the initial ranking be $\boldsymbol{\pi} = \begin{pmatrix}1 & 2 & \dots & n\end{pmatrix}$, where team 1 is ranked first, team 2 is ranked second and so on. Furthermore, in each iteration we will only allow the new permutation to be a transposition away from the permutation in the preceeding iteration. From this we will move on to allowing new permutations that are 3-cycles away from the preceeding permutation. We will then make some alterations to these ideas, and even write a program that checks all possible permutations. We end this section with a comparison on all these different instances of the program that aims to solve Problem 5.18.

5.3.1 New permutations are transpositions (2-cycles) away from preceding permutations

We start by only allowing each new 'nearby' permutation (at line 6 in Algorithm 1) to be a transposition away from the previous permutation. This means that for each new permutation we only swap two items in our ranking. The only criterion for this swap is that the corresponding new matrix it results in is closer to hillside form than the previous one. At each iteration we check all possible transpositions and choose the best one. An implementation of Algorithm 1 with this idea may be found in Appendix A.2.4. Note that we take our initial ranking, $\boldsymbol{\pi}$, to be $\boldsymbol{\pi} = \begin{pmatrix}1 & 2 & \dots & n\end{pmatrix}$.

Example 5.19. Assume that $\boldsymbol{\pi_1} = \begin{pmatrix}1 & 2 & 3 & 4\end{pmatrix}$, and that $\boldsymbol{\pi_2} = \begin{pmatrix}1 & 3 & 2 & 4\end{pmatrix}$. Then $\boldsymbol{\pi_2}$ is a transposition away from $\boldsymbol{\pi_1}$. This is quite obvious, however we may use notation from group theory to show this formally.

Let $\boldsymbol{\sigma} = \begin{pmatrix}1 & 2 & 3 & 4\\ 1 & 2 & 3 & 4\end{pmatrix}$ and $\boldsymbol{\tau} = \begin{pmatrix}1 & 2 & 3 & 4\\ 1 & 3 & 2 & 4\end{pmatrix}$. $\boldsymbol{\tau}$ is a transposition. The product of these ($\boldsymbol{\sigma\tau}$) is given (elementwise) by

$$\begin{aligned}(\sigma\tau)(1) &= \sigma(\tau(1)) = \sigma(1) = 1\\ (\sigma\tau)(2) &= \sigma(\tau(2)) = \sigma(3) = 3\\ (\sigma\tau)(3) &= \sigma(\tau(3)) = \sigma(2) = 2\\ (\sigma\tau)(4) &= \sigma(\tau(4)) = \sigma(4) = 4\end{aligned}$$

This results in $\boldsymbol{\pi} = \boldsymbol{\sigma\tau} = \begin{pmatrix}1 & 2 & 3 & 4\\ 1 & 3 & 2 & 4\end{pmatrix}$.

□

Next we check how well the implementation A.2.4 performs on some examples.

Example 5.20. *(One permutation from hillside)* Assume we have the following matrix.

$$\boldsymbol{D} = \begin{pmatrix} 1 & 5 & 3 \\ 0 & 1 & 0 \\ 0 & 4 & 2 \end{pmatrix}$$

The program finds the following hillside form with the permutation $\boldsymbol{\pi} = \begin{pmatrix} 1 & 3 & 2 \end{pmatrix}$,

$$\boldsymbol{D}_{\boldsymbol{\pi}} = \begin{pmatrix} 1 & 3 & 5 \\ 0 & 2 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$

As the program checks all transpositions in each case, another result would have been rather disappointing. □

Example 5.21. *(Pedings et. al.)* The following matrix is the one from the article of Pedings et. al. It has a hidden hillside form, so let us see how close to hillside form the matrix comes with our new method.

$$\boldsymbol{D} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 4 & 0 & 2 \\ 5 & 0 & 0 & 0 & 0 \\ 15 & 3 & 8 & 0 & 5 \\ 6 & 0 & 3 & 0 & 0 \end{pmatrix}$$

We get that $\boldsymbol{\pi} = \begin{pmatrix} 5 & 2 & 3 & 1 & 4 \end{pmatrix}$. The corresponding permuted matrix is this:

$$\boldsymbol{D}_{\boldsymbol{\pi}} = \begin{pmatrix} 0 & 3 & 5 & 8 & 15 \\ 0 & 0 & 2 & 4 & 9 \\ 0 & 0 & 0 & 3 & 6 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

It has distance 0 from hillside form, i.e. $\boldsymbol{D}_{\boldsymbol{\pi}}$ is on hillside form. The program really improved on the original distance from hillside form for the matrix, which was 247. □

It might seem like a bad idea to only look for improving transpositions in each step. But recall that from Corollary 5.17 we know that any permutation in S_n can be written as a product of transpositions. So if a matrix has a hillside form and thus a corresponding permutation that can take it there, this permutation can be written as a product of transpositions.

With this being said, it is of course a possibility that this method will not reach the optimal permutation since we have designed this method to always

choose the best transposition at all times[2]. In other words, the "wrong" transpositions can be chosen, and thus the optimal permutation never reached. However it is possible to modify this idea to sometimes allow a transposition that takes the matrix a little bit further from hillside, and hope that this results in a better solution further down road.

5.3.2 Allowing "bad" choices of transpositions

The method outlined in Section 5.3.1 always choose the transposition that decreases the distance to hillside form the most and stops whenever an improving move cannot be found. However, as with many local search methods this may lead to that the program never actually finds an optimal solution. Instead of a global minimum we may be left with only a local minimum.

One way to solve this potential problem is to allow the program to sometimes choose transpositions that results in a slight increase in the distance from hillside form. Furthermore we only allow this in the instance of the program where it is about terminate. This means that when the program cannot find any improvements on the current permutation/ranking $\boldsymbol{\pi}$, there is a chance that it will take a step back on a seemingly worse path in hope of a better result further down this (possibly) new road.

Since the entire point of this idea is that we *sometimes* will allow a transposition that takes the matrix further from hillside form, we introduce $\alpha \in [0, 1]$. α will be the probability that the program choose a somewhat worse $\boldsymbol{\pi}$ when it otherwise would have been finished. The following pseudocode illustrates this idea. Notice that one needs to choose some value for α. If α is 1 the program will run forever, and if α is 0 the program will be the same as the program in Section 5.3.1. Choosing α too large means a too long runtime, while choosing α too small means a greater chance of getting stuck in a local minimum.

We remark that we do not know of any matrices for which this modification is actually required. None of the examples we have looked at benefit from it, but we include it in this thesis because it is a common pitfall for local methods to get stuck in local minima.

[2]This is an example of a so-called *greedy* algorithm. What the greedy algorithms have in common is that they always choose the locally optimal alternative. Another example of such algorithms is the greedy strategy for the traveling salesman problem. At each stage in the algorithm one chooses the city that is closest to the current city, without looking at the big picture.

Algorithm 2 Minimizing distance to hillside form using transpositions and allowing "bad" choices

```
1: if h(D) == 0 then
2:    break                          % D is in hillside form
3: else
4:    find some initial ranking/permutation π
5:    while h(D) ≠ 0 do
6:       for π' 'nearby' π do
7:          D' = P_{π'}^T D P_{π'}
8:          if h(D') < h(D) then
9:             π = π'
10:            D = D'
11:         end if
12:      end for
13:      if no improving nearby π' was found then
14:         β = a random number between 0 and 1
15:         if β < α then
16:            π = least bad nearby π'
17:            D = P_π^T D P_π
18:         else
19:            break
20:         end if
21:      end if
22:   end while
23: end if
```

Above $\boldsymbol{\pi}'$ is nearby $\boldsymbol{\pi}$ if and only if $\boldsymbol{\pi}' = \boldsymbol{\tau}\boldsymbol{\pi}$ for some transposition $\boldsymbol{\tau}$. One possible implementation of this can be found in Appendix A.2.5, where we have chosen $\alpha = 0.2$.

5.3.3 New permutations are 2- and 3-cycles away from preceding permutations

Now we try to improve our method from Section 5.3.1 (meaning that we disregard the idea from Section 5.3.2) by allowing $\boldsymbol{\pi}'$ to be up to two transpositions (any permutation of three items) away from $\boldsymbol{\pi}$. More formally we allow the new permutations in the program to be either a transposition (2-cycle) or a 3-cycle from the previous permutation. An implementation in MATLAB may be found in Appendix A.2.6.

Note that how our implementation does not check all $\binom{n}{3}$ possible choices, it randomly chooses three teams/indices i, j and k and checks all $3! - 1 = 5$ possible permutations of these three teams. The program then chooses the swap that improves most on the distance from hillside form.

In our implementation we also introduce `tol`, which is how many misses in a row we tolerate before we stop the program. One miss here will be one random choice of three indices, where none of the possible 3-cycles of these indices result in a decrease in the distance from hillside form. In our program we typically let `tol` be either 2 or 3.

Example 5.22. *(3-cycle)* We would like to illustrate what a 3-cycle is and what it does to a permutation. Let us therefore say we have some permutation $\boldsymbol{\sigma}$ given as

$$\boldsymbol{\sigma} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{pmatrix}$$

Furthermore assume that we have the following 3-cycle.

$$\boldsymbol{\tau} = (1, 3, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix}$$

The product of these permutations will be

$$\boldsymbol{\pi} = \boldsymbol{\sigma}\boldsymbol{\tau} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix}$$

In our ranking terminology this means that our "guess" for the ranking $\boldsymbol{\pi}$ is changed. It remains to see if the corresponding permutation matrix will lead to a better hillside distance for the point differential matrix in question. □

Since the implementation here randomly chooses the three teams we use to find better permutations and thus better distances from hillside form, for $n > 3$ there is a risk of getting different results for different runs of the program. In Example 5.23 we look at a case where $n = 6$ and see what the consequences of the fact that we get different outputs for each run of the program are.

Example 5.23. *(Olympics)* We see how the program (Appendix A.2.6) performs on an actual example from sports. During the Summer Olympics in London 2012 the handball tournament for women consisted of 2 groups of 6 teams each. Each team in the same group met each of the other teams (in the same group) one time. We take a closer look at Norway's group and the results there. The other teams in the same group were France, South Korea, Spain, Denmark and Sweden. 6 teams meeting each other once gives $\frac{6 \cdot 5}{2} = 15$ matches. The results from

the matches were the following[3].

Teams	**Result**	**Teams**	**Result**
Spain - South Korea	27 - 31	Spain - Denmark	24 - 21
Denmark - Sweden	21 - 18	South Korea - France	21 - 24
Norway - France	23 - 24	Spain - Sweden	25 - 24
South Korea - Denmark	25 - 24	Denmark - Norway	23 - 24
France - Spain	18 - 18	Sweden - South Korea	28 - 32
Sweden - Norway	21 - 24	Norway - Spain	20 - 25
Norway - South Korea	27 - 27	Denmark - France	24 - 30
France - Sweden	29 - 17		

In handball a team gets 2 points for a victory, 1 point for a draw and 0 points for a loss. Thus the resulting table after the group stage was this:

	Team	**Points**
1	France	9
2	South Korea	7
3	Spain	7
4	Norway	5
5	Denmark	2
6	Sweden	0

The four best teams qualified for quarter finals, which means that Denmark and Sweden were out of the tournament. Norway did well in the quarter finals and won the entire tournament. Now let us check what the results after the group stage are when we use our program. The point differential matrix corresponding to the results in the group stage is this:

$$\boldsymbol{D} = \begin{array}{c} \\ \text{N} \\ \text{F} \\ \text{SK} \\ \text{Sp} \\ \text{D} \\ \text{Sw} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{N} & \text{F} & \text{SK} & \text{Sp} & \text{D} & \text{Sw} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 3 \\ 1 & 0 & 3 & 0 & 6 & 8 \\ 0 & 0 & 0 & 4 & 0 & 4 \\ 5 & 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

[3]http://en.wikipedia.org/wiki/Handball_at_the_2012_Summer_Olympics_%E2%80%93_Women%27s_tournament

Which teams does the program think deserve to play in the quarter finals? We run the program several times and take a look at two different outputs. The first output is this:

```
D_best =

     0     3     0     1     6     8
     0     0     4     0     0     4
     0     0     0     5     3     1
     0     0     0     0     1     3
     0     0     0     0     0     3
     0     0     0     0     0     0


dist =

    42


pi =

     4     1     2     3     5     6
```

We see that the matrix the program found is in distance 42 from hillside form, and that the permutation that gives this result is $\boldsymbol{\pi} = \begin{pmatrix} 4 & 1 & 2 & 3 & 5 & 6 \end{pmatrix}$. This means that team 1 (Norway, this can be seen from the point differential matrix $\boldsymbol{D}$) is ranked as number four, team 2 (France) is ranked as number one, and so on. The resulting table is this:

	Team
1	France
2	South Korea
3	Spain
4	Norway
5	Denmark
6	Sweden

We see that this list is identical with the actual list from the tournament, with the exception of South Korea and Spain which switched places. Since the four first teams proceed to the quarter finals, there is no difference from the actual tournament.

Now we take a look at another run of the program, this gives the following output:

```
D_best =

     0     0     3     6     1     8
     0     0     0     3     5     1
     0     4     0     0     0     4
     0     0     0     0     0     3
```

```
      0     0     0     1     0     3
      0     0     0     0     0     0


dist =

     47


pi =

      5     1     3     2     4     6
```

Note that the program did not find a hillside form for $\boldsymbol{D}$ this run either. The ranking it found is this: $\boldsymbol{\pi} = \begin{pmatrix} 5 & 1 & 3 & 2 & 4 & 6 \end{pmatrix}$. Meaning that team 1 (Norway) is placed fifth, team 2 (France) is placed first and so on. This gives the following table:

	Team
1	France
2	Spain
3	South Korea
4	Denmark
5	Norway
6	Sweden

According to this, Norway would not have reached the quarter finals and could consequently not have won the tournament. □

Since the method implemented in Appendix A.2.6 randomly chooses three indices (teams) among the n possible ones, uniqueness of the output is not guaranteed. As we saw in Example 5.23, this means that we will *not* necessary get the same results every time we run the program. Here one might of course be either lucky or unlucky (or something in between), this might be seen as a flaw for this implementation.

However, by not checking *all* possible permutations we ensure that our program is (fairly) quick and computationally lightweight compared to its brute-forced alternative. That being said, a program that checks all the possible permutations will always find the optimal solution, while a program that randomly chooses 3 teams to swap might possibly not.

A brute-force program, even though probably slower and computationally heavier, would however provide a nice way to check how well the "randomly choose three indices"-implementation and the transposition implementation in Section 5.3.1 performs. Such an implementation could work as a so called benchmark implementation, i.e. a program to measure the other programs performance

by. If the difference between our programs and the brute-force benchmark program is sufficiently small (relatively speaking), then we have a confirmation that our new program performs reasonably well.

On the other hand, if the difference between the optimal solution for the benchmark program and the optimal solution for another implementation is huge (again, relatively speaking), it will be an indication that the non-brute-force programs are practically useless. To be able to determine how good our new programs actually perform is the main motivation for actually implementing such a brute-force program.

5.3.4 Brute-force implementation

As mentioned in the previous paragraph it might be convenient to have a program that actually checks all possible choices of permutation, instead of merely choosing some indices at random. An implementation of this may be found in Appendix A.2.7. What the program essentially does is checking all $n!$ possible permutations of the n items/teams. Among all these possibilities the program then chooses the best option. Best here of course means the choice that takes the matrix closest to hillside form (in terms of distance). We give an overview of this idea in the following pseudocode.

Algorithm 3 Minimizing distance to hillside form (solving Problem (5.18)) by checking all permutations

```
1: distance = h(D)
2: n = length(D)
3: if distance == 0 then
4:    break                        % D is in hillside form
5: else
6:    for each π in S_n do
7:       Calculate D' = (P_π)^T D P_π and h(D')
8:       if h(D') < distance then
9:          distance = h(D')
10:         π_best = π'
11:         if distance == 0 then
12:            break                % hillside form found
13:         end if
14:      end if
15:   end for
16: end if
```

A brute-force implementation like this will always find the optimal solution to the problem. However, the drawback for any brute-force implementation is the computation time. It is well known that $n!$ grows very fast. Say for example we consider a case of a sports league where 10 teams meat each other. This brute-force program could risk to check the total number of $10! = 3628800$ permutations. Therefore, as previously mentioned, this implementation is mostly useful to check the other programs' performance on relatively small examples.

5.3.5 Choosing a better initial ranking $\boldsymbol{\pi}$

So far the only changes we have made to the program regard the search/choice of new permutations $\boldsymbol{\pi}'$ are in the while loop of Algorithm 1. However, we may make other changes to the program as well. One thing we may do is have a more heuristic approach to this problem.

Until now the initial ranking $\boldsymbol{\pi}$ has been the naive variant, namely $\boldsymbol{\pi} = \begin{pmatrix} 1 & 2 & \dots & n \end{pmatrix}$. This means that our initial guess for the ranking is team 1 is ranked first, team 2 is ranked second, and so on, until team n is ranked in nth place. However, there is probably a better starting point than this. Using some criteria that gives an initial ranking that is closer to the final ranking would be nice, as this would (hopefully) result in a better run time for the program.

What could be a criterion that is easily calculated and that also gives a good starting point for the program? Such a criterion could for example be that our initial ranking sorts the teams by descending number of row sums. In sports terminology, this would mean that the first team in the preliminary ranking is the one that has scored the most points when all matches are taken into account (this corresponds to summing the elements in each row). We check what the initial permutation (ranking) results in when we apply this heuristic to an example.

Example 5.24. *(Pedings et. al., row sums)* Consider the point differential matrix from [PLY12] considered in Example 5.21. We check what the row sums are in this case, and which $\boldsymbol{\pi}$ we get when we sort by descending row sums. First we find the row sums:

$$\begin{aligned}
&\text{Row 1: } 0+0+0+0+0 = 0\\
&\text{Row 2: } 9+0+4+0+2 = 15\\
&\text{Row 3: } 5+0+0+0+0 = 5\\
&\text{Row 4: } 15+3+8+0+5 = 31\\
&\text{Row 5: } 6+0+3+0+0 = 9
\end{aligned}$$

Sorting the teams by descending row sums we get: team 4 (31 points), team 2 (15 points), team 5 (9 points), team 3 (5 points) and lastly team 1 (0 points). In our $\boldsymbol{\pi}$-notation this yields the following ranking

$$\boldsymbol{\pi} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 2 & 4 & 1 & 3 \end{pmatrix} = \begin{pmatrix} 5 & 2 & 4 & 1 & 3 \end{pmatrix}$$

This gives rise to the following permutation matrix (according to Definition 5.11)

$$\boldsymbol{P_\pi} = \begin{pmatrix} \boldsymbol{e_{\pi(1)}} \\ \boldsymbol{e_{\pi(2)}} \\ \boldsymbol{e_{\pi(3)}} \\ \boldsymbol{e_{\pi(4)}} \\ \boldsymbol{e_{\pi(5)}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{e_5} \\ \boldsymbol{e_2} \\ \boldsymbol{e_4} \\ \boldsymbol{e_1} \\ \boldsymbol{e_3} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Then our updated $\boldsymbol{D_\pi}$ is

$$\boldsymbol{D_\pi} = \boldsymbol{P_\pi}^T \boldsymbol{D} \boldsymbol{P_\pi} = \begin{pmatrix} 0 & 3 & 5 & 8 & 15 \\ 0 & 0 & 2 & 4 & 9 \\ 0 & 0 & 0 & 3 & 6 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The matrix $\boldsymbol{D_\pi}$ is in hillside form and we have found our ranking $\boldsymbol{\pi}$. □

Notice that all we did in Example 5.24 was attempting to find a better initial guess for $\boldsymbol{\pi}$, and we ended up finding a $\boldsymbol{\pi}$ such that $\boldsymbol{D_\pi}$ is in hillside form. This is of course a bonus we cannot count on every time, but hopefully by making this educated guess for $\boldsymbol{\pi}$ we may find a better and/or quicker (in terms of computations) solution to Problem 5.18. Alone this is of course not a way of finding an optimal solution to Problem 5.18, but can be used together with for example the transposition implementantion in Section 5.3.1, possibly lowering the computation time and possibly also improving the solution.

We mention here something we will treat in more detail in Chapter 6, that is the reason why this (sorting on row sums) is a good approach for matrices that actually have hidden hillside forms. By sorting on row sums in Example 5.24 we get the same ranking as we would get if we sorted on the number of nonzero entries in each row. This is the same ranking we would get as if we topologically ordered the adjacency graph of the matrix. We return to this topic with formal definitions in Chapter 6.

In Example 5.24 all the row sums were distinct, so sorting them is straightforward, likewise the programming of this procedure. However, it is not always the case that all the row sums are distinct. Some criteria might therefore be needed, to resolve any ties in the row sums.

One natural criteria might be the following: if the sum of row number i and of row number j are equal we decide their initial ranking by assuming that the best of the two teams are the winning team from the match team i and team j played each other. A suggested implementation of this idea, where the problem of two equal row sums is taken into account, can be found in Appendix A.2.8.

5.4 One way to compare rankings

In the two next sections we will compare the different ranking methods (i.e. implementations) we have seen in Chapter 4 and in this chapter on some examples. If the rankings we get from the different methods are identical, then we have no problems. However, when we get rankings that are not identical, how shall we measure how different or how far from each other they are? There are of course many possible ways to answer this, but here we will use what is known as the *Spearman footrule distance*. We use the same definition as they do in [DM10], but we notice that the distance function we introduce is the ℓ_1-norm for vectors in $\mathbb{R}^n$ written in terms of rankings.

Definition 5.25. Given two rankings, $\boldsymbol{\pi}$ and $\boldsymbol{\sigma}$, on n items (i.e. $\boldsymbol{\pi}, \boldsymbol{\sigma} \in S_n$). Then the distance between these rankings is given by the function $d : S_n \times S_n \to \mathbb{R}_+$ (where $\mathbb{R}_+ = [0, \infty)$), where

$$d(\boldsymbol{\pi}, \boldsymbol{\sigma}) = \sum_{i=1}^{n} |\boldsymbol{\pi}(i) - \sigma(i)|$$

Example 5.26. Assume we have the following rankings in S_5

$$\boldsymbol{\pi} = \begin{pmatrix} 1 & 3 & 2 & 5 & 4 \end{pmatrix}$$

$$\boldsymbol{\sigma} = \begin{pmatrix} 4 & 3 & 5 & 1 & 2 \end{pmatrix}$$

Then these rankings are in distance

$$d(\boldsymbol{\pi}, \boldsymbol{\sigma}) = \sum_{i=1}^{5} |\boldsymbol{\pi}(i) - \boldsymbol{\sigma}(i)| = |1-4| + |3-3| + |2-5| + |5-1| + |4-2| = 12$$

from eachother. □

We remark that the distance between any two rankings $\boldsymbol{\pi}$ and $\boldsymbol{\sigma}$ can be calculated in MATLAB with the call `norm(`$\pi - \sigma$`, 1)`, where the 1 reflects the fact that it is indeed the ℓ_1-norm we use.

5.5 Comparisons of the different implementations

Our concern in the previous subsections has been to create and improve a program that take a given point differential matrix to hillside form or as close to hillside form as it gets. Now we will compare these different implementations to eachother and to the brute-force implementation described in the preceeding subsection to see how well the different implementations actually performs[4].

Notice that before we introduce a way of choosing a smarter initial ranking π, the initial ranking is assumed to be $\boldsymbol{\pi} = \begin{pmatrix} 1 & 2 & \dots & n \end{pmatrix}$. After we choose a better $\boldsymbol{\pi}$, we rerun the examples for one of the programs to see if the results improve. Note also that for the program from Section 5.3.3 we let `tol` $= 2$. When we report the ℓ_1-distance it is always the distance between the current ranking and the brute-force found ranking for the same example.

Implementation	Distance from hillside form	Ranking	ℓ_1-distance from brute-force
2-cycles (5.3.1)	0	$\boldsymbol{\pi} = \begin{pmatrix} 5 & 2 & 4 & 1 & 3 \end{pmatrix}$	0
2-cycles, $\alpha = 0.2$ (5.3.2)	0	$\boldsymbol{\pi} = \begin{pmatrix} 5 & 2 & 4 & 1 & 3 \end{pmatrix}$	0
2- and 3-cycles (5.3.3)	49	$\boldsymbol{\pi} = \begin{pmatrix} 4 & 2 & 3 & 1 & 5 \end{pmatrix}$	4
Choose better $\boldsymbol{\pi}$ (5.3.5)	0	$\boldsymbol{\pi} = \begin{pmatrix} 5 & 2 & 4 & 1 & 3 \end{pmatrix}$	0
2- and 3-cycles (new $\boldsymbol{\pi}$)	0	$\boldsymbol{\pi} = \begin{pmatrix} 5 & 2 & 4 & 1 & 3 \end{pmatrix}$	0
Brute-force (5.3.4)	0	$\boldsymbol{\pi} = \begin{pmatrix} 5 & 2 & 4 & 1 & 3 \end{pmatrix}$	0

Table 5.1: *Example 5.21*

[4]Bear in mind that the implementation in Section 5.3.3 randomly chooses three indices, so there will be several possible outputs in this case. We only report one of these in the tables 5.1 and 5.2

Implementation	Distance from hillside form	Ranking	ℓ_1-distance from brute-force
2-cycles (5.3.1)	42	$\boldsymbol{\pi} = \begin{pmatrix} 4 & 1 & 3 & 2 & 5 & 6 \end{pmatrix}$	0
2-cycles, $\alpha = 0.2$ (5.3.2)	42	$\boldsymbol{\pi} = \begin{pmatrix} 4 & 1 & 3 & 2 & 5 & 6 \end{pmatrix}$	0
2- and 3-cycles (5.3.3)	55	$\boldsymbol{\pi} = \begin{pmatrix} 4 & 2 & 3 & 1 & 5 & 6 \end{pmatrix}$	2
Choose better $\boldsymbol{\pi}$ (5.3.5)	42	$\boldsymbol{\pi} = \begin{pmatrix} 4 & 1 & 3 & 2 & 5 & 6 \end{pmatrix}$	0
2- and 3-cycles (new $\boldsymbol{\pi}$)	42	$\boldsymbol{\pi} = \begin{pmatrix} 4 & 1 & 3 & 2 & 5 & 6 \end{pmatrix}$	0
Brute-force (5.3.4)	42	$\boldsymbol{\pi} = \begin{pmatrix} 4 & 1 & 3 & 2 & 5 & 6 \end{pmatrix}$	0

Table 5.2: *Example 5.23*

So far our examples have been of rather small dimensions, so now we are going to take a look at a somewhat different example where $n = 16$. This example comes from electronic sports (also known as *esports*[5]).

Example 5.27. This example comes from the group play in a championship tournament known as "The International" for the computer game *Defense of the Ancients 2* (abbreviated to DotA (2)). The group stage is played as round robin, meaning that all teams play each other one time. DotA is a team game with five players on each team. In each match two teams face each other, and the goal is to conquer the other team's base. Each player pick one of several possible heroes (really a fantasy figure with some special skills) before each match. During the game the teams fight each other, while collecting money and experience.

Below we have the point differential matrix for the group stage in "The International 2014". Here the entry D_{ij} will be 0 if team j beat team i, and otherwise be the number of points (i.e. money per minute) that team i beat team j by[6].

[5]If you immeadiately think esports are not sports then bear in mind that bananas really are berries, botanically speaking.

[6]http://wiki.teamliquid.net/dota2/The_International/2014/Playoffs/Phase_Two

$$\boldsymbol{D} = \begin{pmatrix}
0 & 637 & 0 & 0 & 0 & 0 & 0 & 453 & 890 & 643 & 1144 & 453 & 0 & 569 & 463 & 0 \\
0 & 0 & 1060 & 188 & 0 & 793 & 422 & 0 & 1082 & 288 & 343 & 875 & 478 & 148 & 0 & 770 \\
368 & 0 & 0 & 0 & 0 & 361 & 541 & 0 & 786 & 527 & 804 & 0 & 0 & 0 & 679 & 0 \\
507 & 0 & 800 & 0 & 0 & 0 & 519 & 406 & 1166 & 0 & 0 & 0 & 1058 & 877 & 0 & 0 \\
758 & 700 & 721 & 585 & 0 & 0 & 511 & 0 & 370 & 50 & 0 & 0 & 499 & 786 & 0 & 460 \\
574 & 0 & 0 & 198 & 318 & 0 & 296 & 0 & 601 & 0 & 0 & 0 & 0 & 0 & 0 & 954 \\
557 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 781 & 559 & 0 & 0 & 787 & 537 & 0 & 218 \\
0 & 304 & 486 & 0 & 259 & 872 & 738 & 0 & 1090 & 0 & 462 & 0 & 707 & 0 & 0 & 930 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1081 & 515 & 0 \\
0 & 0 & 0 & 343 & 0 & 513 & 0 & 676 & 499 & 0 & 0 & 0 & 1008 & 583 & 0 & 0 \\
0 & 0 & 0 & 374 & 531 & 1119 & 603 & 0 & 472 & 383 & 0 & 0 & 0 & 616 & 0 & 296 \\
0 & 0 & 763 & 1044 & 681 & 475 & 1448 & 331 & 383 & 525 & 705 & 0 & 742 & 490 & 810 & 0 \\
371 & 0 & 489 & 0 & 0 & 1 & 0 & 0 & 707 & 0 & 656 & 0 & 0 & 270 & 831 & 0 \\
0 & 0 & 508 & 0 & 0 & 885 & 0 & 67 & 0 & 0 & 0 & 0 & 0 & 0 & 752 & 741 \\
0 & 522 & 0 & 296 & 405 & 1200 & 637 & 545 & 0 & 842 & 429 & 0 & 0 & 0 & 0 & 0 \\
304 & 0 & 940 & 613 & 0 & 0 & 0 & 0 & 671 & 432 & 0 & 700 & 593 & 0 & 23 & 0
\end{pmatrix}$$

The original distance from hillside form for this matrix is 642318. Our results for the different implementations are the following. Note that $n = 16$ is too large for MATLAB's `perms` to handle, hence we have no brute-force solution for this example. In the following table $h(\boldsymbol{D}_{\boldsymbol{\pi}})$ is distance from hillside form for the matrix symmetrically reordered according to the permutation $\boldsymbol{\pi}$.

Implementation	$h(\boldsymbol{D}_{\boldsymbol{\pi}})$	**Ranking, $\boldsymbol{\pi}$**
2-cycles (5.3.1)	447613	$\begin{pmatrix}5 & 2 & 11 & 6 & 4 & 15 & 12 & 3 & 16 & 10 & 9 & 1 & 13 & 14 & 7 & 8\end{pmatrix}$
2-cycles, $\alpha = 0.2$ (5.3.2)	447613	$\begin{pmatrix}5 & 2 & 11 & 6 & 4 & 15 & 12 & 3 & 16 & 10 & 9 & 1 & 13 & 14 & 7 & 8\end{pmatrix}$
2- and 3-cycles (5.3.3)	543810	$\begin{pmatrix}2 & 1 & 10 & 6 & 5 & 13 & 7 & 8 & 9 & 15 & 16 & 3 & 11 & 14 & 4 & 12\end{pmatrix}$
Choose better $\boldsymbol{\pi}$ (5.3.5)	448864	$\begin{pmatrix}6 & 2 & 10 & 5 & 4 & 15 & 12 & 3 & 16 & 11 & 8 & 1 & 13 & 14 & 7 & 9\end{pmatrix}$
2- and 3-cycles (new $\boldsymbol{\pi}$)	448743	$\begin{pmatrix}5 & 2 & 10 & 6 & 4 & 15 & 12 & 3 & 16 & 11 & 8 & 1 & 13 & 14 & 7 & 9\end{pmatrix}$
Brute-force (5.3.4)	n/a	n/a

□

Since the point differential matrix in Example 5.27 contains a lot more entries and also larger numbers than the examples we have seen so far, it is not surprising that the distance from hillside form for this matrix is a lot larger than for the rest of the examples. However, even though the situations are different we may be able to compare these examples. This can be done by scaling the numbers. We could for example introduce a scaled distance from hillside form, and defined it to be

$$h^*(\boldsymbol{D}) = \frac{h(\boldsymbol{D})}{\sum_{i=1}^{n}\sum_{j=1}^{n} D_{ij}},$$

where $h(\boldsymbol{D})$ is the original distance from hillside form in Definition 5.3. Since the rest of the examples in this thesis will be of roughly the same magnitude, we will not dwell more on this scaling here.

5.6 Comparing the two hillside methods on some examples

In the current and the previous chapter we have seen two different ranking methods, both trying to take a point differential matrix to hillside form. In this chapter we will compare these two methods on various examples.

In Chapter 4 we looked at the method in the article [PLY12], and in Section 4.3 we even implemented the method of Pedings et. al. in OPL. Since no code is given in the article [PLY12], our OPL-implementation of the BILP is what we will use from here. Earlier in this chapter we created a minimum distance ranking method. Both methods start with a point differential matrix for n items, and tries to find a hillside form for that matrix.

A natural question to ask now is: how do the two methods perform on the same examples? In this section we will compare the rankings these two methods produce on some examples.

Earlier in Chapter 5 we made different implementations for the different algorithms for the minimum distance method. We will use only one of these implementations here, that is the one from Section 5.3.3 with the educated guess for $\boldsymbol{\pi}$ (i.e. where the initial $\boldsymbol{\pi}$ is based on sorting the items by descending row sums in the point differential matrix).

We start by considering matrices with hidden hillside forms.

$$\boldsymbol{A_1} = \begin{pmatrix} 0 & 3 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 14 & 17 & 0 & 8 & 18 & 7 & 3 \\ 1 & 4 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 5 & 0 & 1 & 10 & 0 & 0 \\ 10 & 15 & 0 & 7 & 17 & 6 & 0 \end{pmatrix} \quad \boldsymbol{A_2} = \begin{pmatrix} 0 & 15 & 0 & 18 & 4 & 0 & 2 & 11 & 13 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 7 & 16 & 0 & 22 & 11 & 0 & 10 & 14 & 15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 9 & 0 & 0 & 0 & 5 & 7 \\ 9 & 20 & 3 & 23 & 15 & 0 & 11 & 16 & 19 \\ 0 & 12 & 0 & 15 & 1 & 0 & 0 & 10 & 11 \\ 0 & 5 & 0 & 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 8 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

Both $\boldsymbol{A_1}$ and $\boldsymbol{A_2}$ have hidden hillside forms, and in both cases both methods (i.e. implementations) returns the same ranking. Those rankings yield a hillside

form for both matrices. For the sake of clarity, these are $\boldsymbol{\pi_1} = \begin{pmatrix}5 & 6 & 1 & 4 & 7 & 3 & 2\end{pmatrix}$ and $\boldsymbol{\pi_2} = \begin{pmatrix}3 & 8 & 2 & 9 & 5 & 1 & 4 & 6 & 7\end{pmatrix}$, for $\boldsymbol{A_1}$ and $\boldsymbol{A_2}$ respectively.

Now we take a look at matrices without hidden hillside forms, and see if the rankings we get are different in those cases. The following matrices do not have hidden hillside forms. Note that in the tables 5.3, 5.4 and 5.5 we also list the differences between the rankings that the methods produce; the ℓ_1-norm between the permutations, and the absolute value of the difference of both the distance from hillside form and the number of violations from hillside form.

$$\boldsymbol{B_1} = \begin{pmatrix}
0 & 7 & 11 & 0 & 3 & 0 & 1 & 3 & 1 & 6 \\
13 & 10 & 0 & 9 & 9 & 19 & 0 & 0 & 0 & 8 \\
0 & 16 & 15 & 0 & 0 & 18 & 0 & 17 & 18 & 0 \\
19 & 8 & 6 & 0 & 1 & 8 & 0 & 0 & 17 & 1 \\
12 & 0 & 0 & 7 & 0 & 17 & 3 & 0 & 0 & 17 \\
0 & 0 & 0 & 4 & 0 & 0 & 9 & 0 & 0 & 6 \\
0 & 8 & 15 & 10 & 17 & 0 & 12 & 0 & 0 & 0 \\
19 & 0 & 4 & 11 & 20 & 0 & 0 & 12 & 0 & 0 \\
0 & 8 & 9 & 0 & 8 & 13 & 4 & 0 & 0 & 18 \\
0 & 15 & 14 & 18 & 16 & 0 & 0 & 7 & 20 & 0
\end{pmatrix}$$

$$\boldsymbol{B_2} = \begin{pmatrix}
0 & 6 & 0 & 5 & 0 & 8 & 0 \\
0 & 0 & 0 & 0 & 0 & 5 & 4 \\
8 & 14 & 0 & 0 & 7 & 17 & 10 \\
0 & 5 & 3 & 0 & 0 & 7 & 0 \\
10 & 13 & 0 & 14 & 0 & 15 & 8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 7 & 0 & 9 & 0
\end{pmatrix} \qquad \boldsymbol{B_3} = \begin{pmatrix}
0 & 14 & 0 & 0 & 15 & 11 & 17 & 10 \\
0 & 0 & 0 & 0 & 4 & 0 & 9 & 0 \\
14 & 21 & 0 & 9 & 10 & 19 & 23 & 16 \\
15 & 0 & 0 & 0 & 19 & 18 & 20 & 15 \\
0 & 0 & 0 & 3 & 0 & 4 & 5 & 0 \\
0 & 7 & 0 & 0 & 0 & 0 & 10 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 8 & 0 & 0 & 13 & 9 & 16 & 0
\end{pmatrix}$$

Implementation	Ranking	Distance from hillside form	Number of violations from hillside form
Min. violations	$\boldsymbol{\pi} = \begin{pmatrix}8 & 4 & 5 & 7 & 9 & 10 & 2 & 1 & 6 & 3\end{pmatrix}$	2672	298
Min. distance	$\boldsymbol{\pi} = \begin{pmatrix}9 & 7 & 2 & 6 & 8 & 10 & 3 & 4 & 5 & 1\end{pmatrix}$	2658	314
(Differences)	16	14	16

Table 5.3: For the matrix $\boldsymbol{B_1}$.

Implementation	Ranking	Distance from hillside form	Number of violations from hillside form
Min. violations	$\boldsymbol{\pi} = \begin{pmatrix} 4 & 6 & 1 & 5 & 2 & 7 & 3 \end{pmatrix}$	126	27
Min. distance	$\boldsymbol{\pi} = \begin{pmatrix} 4 & 6 & 2 & 5 & 1 & 7 & 3 \end{pmatrix}$	126	29
(Differences)	2	0	2

Table 5.4: For the matrix $\boldsymbol{B_2}$.

Implementation	Ranking	Distance from hillside form	Number of violations from hillside form
Min. violations	$\boldsymbol{\pi} = \begin{pmatrix} 3 & 6 & 1 & 2 & 7 & 5 & 8 & 4 \end{pmatrix}$	180	30
Min. distance	$\boldsymbol{\pi} = \begin{pmatrix} 3 & 5 & 1 & 2 & 7 & 6 & 8 & 4 \end{pmatrix}$	173	31
(Differences)	2	7	1

Table 5.5: For the matrix $\boldsymbol{B_3}$.

Now we have seen the two methods applied to some examples. For the examples with a hidden hillside form, the methods produced the same ranking, but for the matrices without a hidden hillside form they produced slightly different rankings. From the example with largest n, $\boldsymbol{B_1}$, it can seem like the difference in the rankings is proportional to the size of the matrix, n.

However, in the cases $\boldsymbol{B_2}$ and $\boldsymbol{B_3}$ the differences between the rankings are small, and in all three cases each of the methods performs best at what they were designed for (either minimizing distance or minimizing the number of violations). We will not dwell more on the comparison of these methods here, but bear in mind that the methods return relatively similar rankings and that the differences that do occur are due to the different goals of the two methods.

Chapter 6

The hillside form and graphs

The intention of this chapter is to take a look at the graph theoretical aspects of the hillside form for matrices, in the hope that this will shed some light over what kind of matrices that have hidden hillside forms. And, perhaps more important, show a different side of the problem of finding a permutation $\boldsymbol{\pi}$ such that $\boldsymbol{P_\pi}^T \boldsymbol{D} \boldsymbol{P_\pi}$ is closest to hillside form (either in terms of violations or distance).

In other words, we will look at the underlying graphs of the matrices we want to find a ranking for. To get there we start by making some definitions, for which our main source is [BM76], and for sections 6.2 and 6.3 we turn to Chapter 3 of [Saa03] and to lecture notes by James R. Lee ([Lee]). Towards the end of this chapter we show that we can find the hillside form (if it exists) of a matrix by finding a topological ordering of the adjacency graph of the matrix.

6.1 A quick introduction to graph theory

First we list some basic definitions from graph theory that we will need later.

A *graph* G is an ordered pair $G = (V, E)$ consisting of a set of vertices (also called nodes) $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges (also known as arcs) $E = \{e_1, e_2, \ldots, e_m\}$ between the vertices. If the edges are ordered pairs, i.e. that $e_i = (v_j, v_k)$ (the edge "goes from" vertex v_j to vertex v_k), we say that G is a *directed graph* or a just a *digraph*. On the other hand, if each edges of G is unordered, i.e. that $e_i = \{v_j, v_k\}$, there simply is an edge between v_j and v_k without any direction.

A graph may be visualised as points in the plane for vertices and lines between these points representing the edges. Multiple examples of this can be seen

in sections 6.2 and 6.3.

A graph $G = (V, E)$ is *weighted* if to each edge $e_i \in E$ there is a corresponding weight w_i (typically $w_i \in \mathbb{N}$ or $w_i \in \mathbb{R}$), meaning that we have a set of weights $W = \{w_1, w_2, \ldots, w_m\}$.

A *path* in a graph $G = (V, E)$ is an ordered sequence of vertices where no vertex appears twice and between any two neighbour vertices in the sequence there exists an edge in the graph.

If we have a graph $G = (V, E)$, and there exist paths between any pair of vertices in G, we say that G is a *connected* graph. Furthermore, if G is a directed graph, we say that G is *strongly connected* if for each pair of vertices (v_j, v_k) there is a path of directed edges from v_j to v_k. A maximal subset of vertices that can be connected by paths in a graph is known as a *connected component* of the graph.

A cycle in a graph means an ordered sequence of vertices and edges that comes back to the first vertex in the sequence, meaning that the first and last vertex in the sequence is the same vertex. A graph without any cycles is called an *acyclic* graph. A *cyclic* graph is a graph containing one or more cycles.

A *loop* in a graph is an edge with the same vertex at both ends (the head and tail of the edge are the same), i.e. that $e_i = (v_j, v_j)$ for some vertex v_j.

For the rest of the chapter we make the assumption that the graphs we consider are without loops and also that there is a maximum of one edge between any two vertices.

Let G and H be two graphs, and denote their vertex and edge sets as $V(G)$ and $V(H)$, respectively. A bijection $\phi : V(G) \to V(H)$ between the vertex sets of G and H is a *graph isomorphism* when any two vertices u and v of G are adjacent in G if and only if $\phi(u)$ and $\phi(v)$ are adjacent in H. We then say that G and H are isomorphic, and denote this by $G \simeq H$. If we have $G \simeq G$ (like in our case where $\pi : V(G) \to V(G)$), we say that the bijection is a graph automorphism.

6.2 Graphs and matrices

Why are we suddenly so interested in graphs, when we have been concerned with matrices in the majority of this thesis? This is of course because the concepts of graphs and matrices are strongly connected. We introduce now what is known

as adjacency matrices. The name comes from the fact that we call two vertices *adjacent* if there is an edge between them.

Definition 6.1. Let $G = (V, E)$ be an unweighted graph, where $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_n\}$. Then we define the *adjacency matrix* of G to be the $n \times n$ $(0,1)$-matrix $\boldsymbol{A} = [a_{ij}]_{i,j=1}^{n}$ with entries defined by

$$a_{ij} = \begin{cases} 1 & \text{if there is an edge from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

If G is undirected it follows that $\boldsymbol{A} = \boldsymbol{A}^T$, i.e. that $\boldsymbol{A}$ is symmetric.

The above definition only considers the unweighted case, but the case of a weighted graph is very similar. Let $W = \{w_1, w_2, \ldots, w_m\}$ be the weights corresponding to the edges $E = \{e_1, e_2, \ldots, e_m\}$, so that w_i is the weight of edge e_i, for $i = 1, 2, \ldots, m$. Then the adjacency matrix $\boldsymbol{A}$ no longer is a $(0,1)$-matrix. There are zeros in all entries a_{ij} where there are no edge from v_i to v_j. But in the cases where there is an edge from v_i to v_j, then $a_{i,j}$ is the weight w_k corresponding to the edge e_k between v_i and v_j.

In the previous paragraph we went from a graph to an adjacency matrix, but of course we can go the other way around. We can start with a matrix $\boldsymbol{A}$ and simply create the corresponding graph as if $\boldsymbol{A}$ was an adjacency matrix. It is only a matter of using Definition 6.1 backwards. Similarly to the naming of adjacency matrices, we call such graphs for *adjacency graphs*. For both adjacency graphs and matrices we sometime drop the word adjacency when no misunderstandings will arise.

One operation that is central to both the MVR method in Chapter 4 and to the ranking method we developed in Chapter 5, is that of symmetric permutations (symmetric reorderings). Recall that to symmetrically permute a matrix $\boldsymbol{D}$, we must apply the same permutation to both the columns and the rows of $\boldsymbol{D}$.

In other words, we get that a symmetric permutation of the $n \times n$-matrix $\boldsymbol{D}$ can be written as $\boldsymbol{P_\pi}^T \boldsymbol{D} \boldsymbol{P_\pi}$, where $\boldsymbol{P_\pi}$ is the permutation matrix corresponding to the permutation $\boldsymbol{\pi}$. So far we have only looked at what happens with the matrix when we do a symmetric permutation, but now we will see what happens to the underlying graph when we take $\boldsymbol{D}$ to be an adjacency matrix.

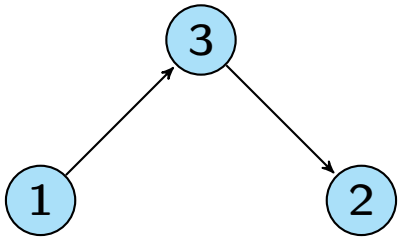
Assume that $\boldsymbol{D}$ is an $n \times n$-matrix, and that $\boldsymbol{D^\pi}$ is the matrix we get when $\boldsymbol{D}$ is symmetrically permuted by the permutation $\boldsymbol{\pi}$. Furthermore, let (i, j) be

an edge in the adjacency graph of $\boldsymbol{D}$ (meaning there is an edge between vertex i and j). Then we have that $\boldsymbol{D}^{\boldsymbol{\pi}}_{i,j} = \boldsymbol{D}_{\boldsymbol{\pi}(i),\boldsymbol{\pi}(j)}$, which means that (i,j) is an edge in the adjacency graph of $\boldsymbol{D}$ if and only if $(\boldsymbol{\pi}(i), \boldsymbol{\pi}(j))$ is an edge in the adjacency graph of $\boldsymbol{D}^{\boldsymbol{\pi}}$.

So, what does it mean for the graph $G_{\boldsymbol{D}}$ when one symmetrically permutes its adjacency matrix $\boldsymbol{D}$? In essence, a symmetric permutation of the matrix by some permutation $\boldsymbol{\pi}$ relabels the vertices in the graph by the same permutation $\boldsymbol{\pi}$. As we saw in the previous paragraph, (i,j) is an edge in $G_{\boldsymbol{D}}$ if and only if $(\boldsymbol{\pi}(i), \boldsymbol{\pi}(j))$ is an edge in $G_{\boldsymbol{D}^{\boldsymbol{\pi}}}$.

This means that the graph "looks" the same after the symmetric permutation, since it has (in some sense) the same edges as before. The only difference in $G_{\boldsymbol{D}^{\pi}}$ is that the names (i.e. the labeling) of the vertices is different. This fact is easily visualised, for example via Example 6.2. For the sake of clarity we mention that the reason why the graph "looks the same" after we have applied $\boldsymbol{\pi}$ to it is because $\boldsymbol{\pi}$ is a graph automorphism.

Example 6.2. Let $G = (V, E)$ be an unweighted graph with vertices $V = \{1, 2, 3\}$ and edges $E = \{(1,3), (3,2)\}$.



G has the following adjacency matrix

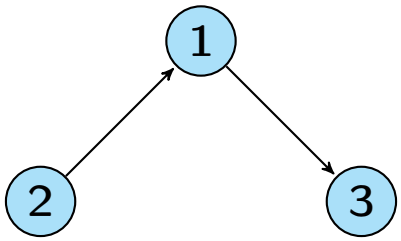
$$\boldsymbol{D_G} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

Now, assume we have the permutation $\boldsymbol{\pi} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$. We check what

happens to G when we symmetrically permute $\boldsymbol{D_G}$ by $\boldsymbol{\pi}$.

$$\boldsymbol{D_G}^{\pi} = \boldsymbol{P_\pi}^T \boldsymbol{D} \boldsymbol{P_\pi} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

This means that the adjacency graph of $\boldsymbol{D_G}^{\pi}$, G_π, is the following.



We see that $G_{\boldsymbol{\pi}} = (V, E')$, where $E' = \{(2,1),(1,3)\} = \{(\boldsymbol{\pi}(1), \boldsymbol{\pi}(3)), (\boldsymbol{\pi}(3), \boldsymbol{\pi}(2))\}$. As we see, this agrees well with the interpretation that symmetrical permutations are relabelings of the vertices, but that the graph really is the same.

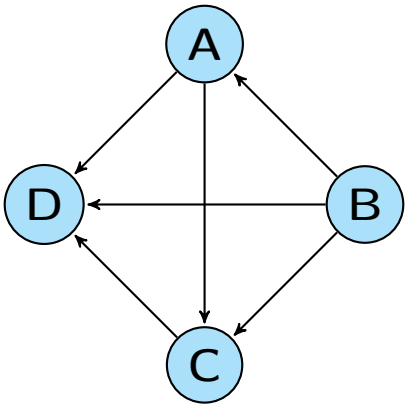
□

6.3 DAGs and topological orderings

If we put together two definitions from Section 6.1 we get a type of graphs that will be of importance to us in this chapter.

Definition 6.3. A graph that is both acyclic and directed is called a *directed acyclic graph* and is abbreviated as DAG. Since we consider here a directed graph, acyclic means of course that the graph contains no *directed cycles*.

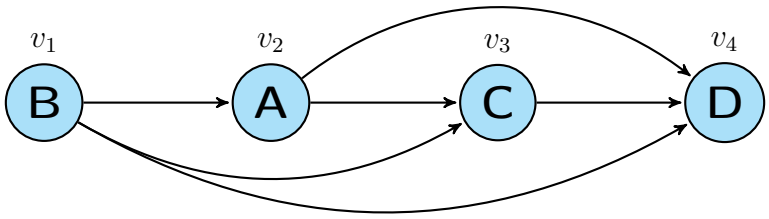
Example 6.4. Here we have a DAG with four vertices.



Definition 6.5. A *topological order* of a directed graph $G = (V, E)$ is an ordering of its vertices as $v_1, v_2, \ldots, v_n$ so that for every edge (v_i, v_j) we have that $i < j$.

Visually we can interpret this as a graph where the vertices are lined up in a queue where no vertex has an incoming edge from a vertex placed to the right of itself.

Example 6.6. Recall our situation in Example 6.4. That graph has a topological ordering if we let $v_1 = B$, $v_2 = A$, $v_3 = C$ and $v_4 = D$. When we draw this graph ordered below, we see that indeed all pairs (v_i, v_j), $i, j = 1, 2, 3, 4$, satisfies that $i < j$.



Notice also that when we have a topological ordering and draw it in that order like we did above, we have no arrows (no directed edges) that goes from right to left in the graph. □

The next question now is this: can we determine when a given graph $G = (V, E)$ is a DAG? It turns out that this is tightly linked to our newly defined concept of topological orderings.

Lemma 6.7. If a graph $G = (V, E)$ has a topological order, then G is a DAG.

Proof. We will prove this by contradiction. Suppose that the graph $G = (V, E)$ has a topological order, $v_1, v_2, \ldots, v_n$. Assume furthermore that G has a directed cycle $C = \{v_{k_1}, \ldots, v_{k_s}\}$. Let v_i be the lowest-indexed vertex in the cycle, and let v_j be the vertex just before v_i in the cycle. This means that (v_j, v_i) is an edge in the graph. Since v_i was taken to be the lowest-indexed verice in the cycle, we must have that $i < j$ (or else v_j would have been the lowest-index vertex in the cycle).

But since we also have that $v_1, v_2, \ldots, v_n$ is a topological ordering of G, and we saw that (v_j, v_i) is an edge, we must have that $j < i$, by the definition of a topological ordering. Hence we have that both $i < j$ and $j < i$ and thus a contradiction; the assumption that G has a directed cycle cannot be right. □

The natural question to ask now is whether or not every DAG has a topological ordering. We shall see that they indeed do, but before we show that we establish a fact regarding the incoming edges of the vertices in a DAG. An incoming edge of a vertex v_i is all edges that ends in vertex v_i, namely edges on the form $(*, v_i)$.

Lemma 6.8. If G is a DAG, then G has a vertex with no incoming edges.

Proof. Again we argue by contradiction. Suppose that G is a DAG and assume that every vertex in G has at least one incoming edge. Now, let us take any vertex, say v_i. Since every vertex is assumed to have at least one incoming edge we may follow these edges backwards from v_i.

Say that (one of) the incoming edge(s) of v_i is from v_j, so (v_j, v_i) is an edge. We apply the same logic to vertex v_j. Since v_j has at least one incoming edge, say (v_k, v_j), we walk backwards to v_k.

Continue in this fashion until we have visited a vertex twice. Let the sequence of these vertices be denoted by C. C is a cycle and hence G cannot be a DAG. Our assumption that every vertex in G must have at least one incoming edge must be wrong. □

The result in Lemma 6.8 establish a rather quick check (at least for small graphs) to see if a graph at all has a chance of being a DAG. If all vertices in a graph has incoming edges, the graph can never be a DAG. Of course this does not mean that any graph with a vertex without incoming edges is a DAG.

Lemma 6.9. If G is a DAG, then G has a topological ordering.

Proof. We show this result by induction on n, the number of vertices in the graph. For $n = 1$, this is obviously true. Assume now that the statement is true for $n - 1$, meaning that for a graph with $n - 1$ vertices there exists a topological ordering.

Now we will see that the statement is also true for a DAG G with $n > 1$ vertices. By Lemma 6.8 we can, for any DAG, find a vertex without incoming edges. Assume this is the case for vertex v_i in G. Then we look at G when we exclude v_i from the graph, namely the graph $G \setminus \{v_i\}$. This new graph cannot possibly have any cycles (since removing a vertex not will create any cycles), so it is a DAG.

Now, by the induction hypothesis $G \setminus \{v_i\}$ has a topological ordering (since it has $n - 1$ vertices). To create a topological ordering of G, place v_i first (recall

that we took v_i to have no incoming edges). Then append the vertices of $G \setminus \{v_i\}$ in the their topological order. Hence graphs with n vertices have topological orderings. □

So far, so good. We have now established the existence of a topological order for every DAG. And to know that something exists is of course a nice thing, but we could also be interested in having an explicit way of computing such a topological ordering for a DAG. The algorithm for finding a topological ordering is well known, and we present it here in the following pseudocode. Note that it resembles the technique from the proof of Lemma 6.9.

Algorithm 4 Finding a topological ordering of a graph G

1: Find a vertex v without incoming edges
2: Let v be first in the topological ordering
3: Remove v from G
4: Recursively find a topological order of $G \setminus \{v\}$ and append this order to v

If n is the number of vertices in a graph, and m the number of edges, then this algorithm finds a topological order for the graph in $\mathcal{O}(n + m)$ time[1].

6.4 Hillside form and graphs

Until now we have barely mentioned the hillside form in this chapter. However, now we will try to connect the theory of graphs and topological orderings to that of hillside forms and ranking.

6.4.1 The $(0, 1)$-matrix case

The first thing we must remark is that a topological ordering looks very much like a ranking of the vertices it contains. Meaning that the vertex ordered first is ranked first, the second vertex is ranked second, etc. Initially this makes us happy, until we with horror realise that the graphs considered in Section 6.3 were unweighted, hence their adjacency matrices were $(0, 1)$-matrices. And we know that matrices in hillside form rarely contain only 0s and 1s.

The fundamental property of hillside matrices is indeed that they have descending order down columns and ascending order across rows. Hence a hillside

[1]Proved various places, for example at page 10 of [Lee].

matrix is a more complex case than a $(0,1)$-matrix. We therefore start by investigating the $(0,1)$-matrices and move on to hillside matrices from there.

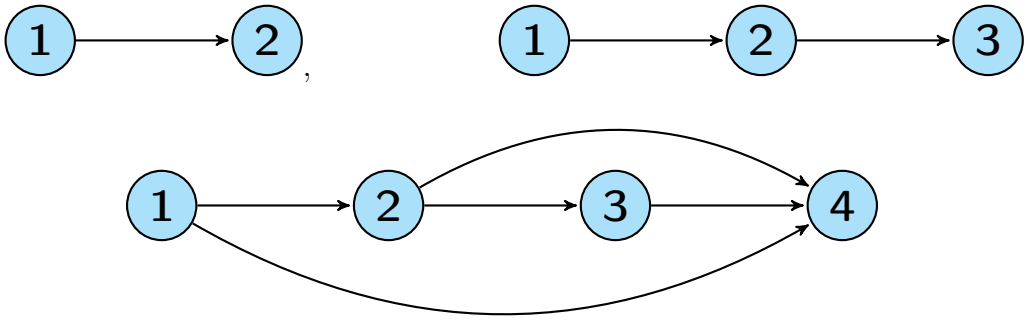
Next we determine when it is possible to symmetrically permute a $(0,1)$-matrix to a strictly upper triangular matrix. In other words, we want to find out when we can find a permutation matrix $\boldsymbol{P_\pi}$ such that

$$\boldsymbol{P_\pi}^T \boldsymbol{A} \boldsymbol{P_\pi} = \begin{pmatrix} 0 & * & * & \dots & * \\ 0 & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & * \\ 0 & \dots & \dots & \dots & 0 \end{pmatrix},$$

where $\boldsymbol{A}$ is a $(0,1)$-matrix and each $*$ is either 0 or 1.

Now, we realize is that any topological ordering has as its adjacency matrix a strictly upper triangular $(0,1)$-matrix. We can of course have 0s in the upper triangular part of the matrix as well, but no 1s in the lower triangular part of the matrix. A 1 in the lower triangular part of the matrix would correspond to an edge going from right to left, and then it would not be a valid topological ordering. In Example 6.10 this fact is verified for the cases $n = 2, 3$ and 4.

Example 6.10. We have the following examples of topological orderings for when n is 2, 3 and 4, respectively.



The corresponding adjacency matrices are these

$$\boldsymbol{A_2} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad \boldsymbol{A_3} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \boldsymbol{A_4} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

□

Since we would like to use some of the aforementioned theory for hillside matrices, the fact that 0s can occur in the upper triangular part of a matrix (like we saw in Example 6.10) is less than great. If a $(0,1)$-matrix was to be in perfect hillside form we would need it to have only 1s in the strict upper triangular part of the matrix[2]. Since these matrices[3] are strict upper triangular $(0,1)$-matrices, they will of course also have topological orderings.

Lemma 6.11. A $(0,1)$-matrix that can be symmetrically permuted to a strict upper triangular matrix has a topological ordering for its adjacency graph.

Proof. Assume that G is the adjacency graph of a $(0,1)$-matrix $\boldsymbol{A}$ that has been symmetrically permuted to a strict upper triangular matrix $\boldsymbol{A'}$. If we line up the vertices of G such that the first vertex in line is the vertex corresponding to the first row of $\boldsymbol{A'}$, the second vertex in line is the one corresponding to the second row in $\boldsymbol{A'}$, and so on. But this is a topological ordering, since no edge goes from the left to the right in the graph. □

Corollary 6.12. A $(0,1)$-matrix that can be symmetrically permuted to a strict upper triangular $(0,1)$-matrix with only 1s in the upper triangular part has a topological ordering for its adjacency graph.

We see that if $\boldsymbol{A}$ is in strictly upper triangular form with only 1s in the upper triangular part of the matrix, then the adjacency graph of $\boldsymbol{A}$ is topologically ordered. From Lemma 6.7 we know that if a graph has a topological order, it is a DAG.

Conversely, if we have a graph that we know is a DAG, we know (from Lemma 6.9) that this graph has a topological ordering. This means that if a graph $G_{\boldsymbol{A}}$ is a DAG, then there exists a permutation $\boldsymbol{\pi}$ such that we may relabel the vertices such that we have a topological ordering, hence a $\boldsymbol{\pi}$ such that ${\boldsymbol{P}_{\boldsymbol{\pi}}}^T \boldsymbol{A} \boldsymbol{P}_{\boldsymbol{\pi}}$ is a stricly upper triangular $(0,1)$-matrix.

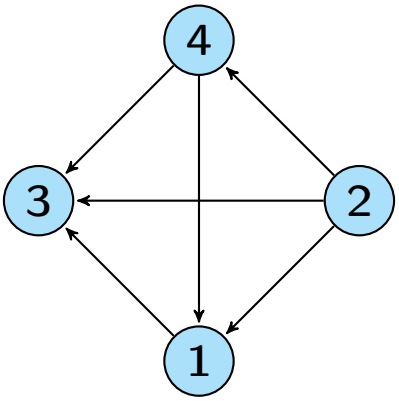
Example 6.13. Let $\boldsymbol{A}$ be the following $(0,1)$-matrix.

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

It has this adjacency graph:

[2]If we allow rows to have equal row sums it would mean that whenever we encountered a 1 in a row, the subsequent entries in the row must be 1s as well. And whenever we encountered a 0 going down a column, the subsequent entries in the column would have to be 0.

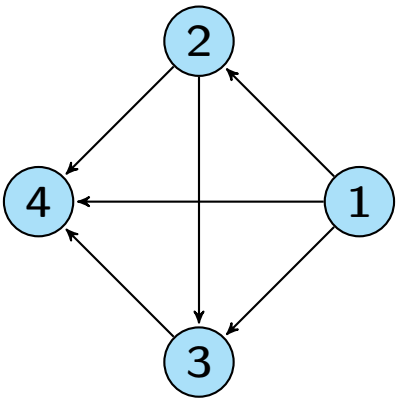
[3]Such matrices arise for example in so called *transitive tournament graphs*.



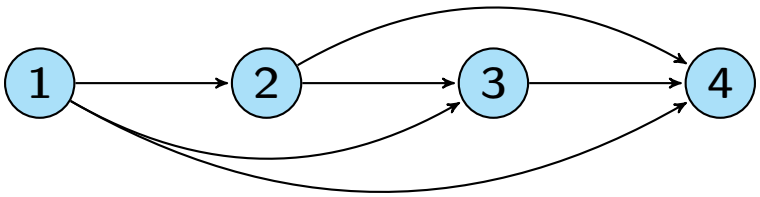
Assume now that we have the permutation $\boldsymbol{\pi} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix}$. We check what happens to $\boldsymbol{A}$ when we symmetrically permutes it by $\boldsymbol{\pi}$.

$$\boldsymbol{A}_{\boldsymbol{\pi}} = {\boldsymbol{P}_{\boldsymbol{\pi}}}^T \boldsymbol{A} \boldsymbol{P}_{\boldsymbol{\pi}} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The corresponding graph is this:



This graph is topologically ordered, as we also can draw the graph as



□

We mention also that the $(0,1)$-matrices are very similar to the matrices we know as decision matrices from Chapter 4, so the fact that these $(0,1)$-matrices may be used for ranking is not very suprising. Here we have merely connected this theory to that of graphs.

Since the matrices we are interested in in this thesis are point differential matrices, the entries (i,j) in our matrices will be 1 if team i beat j by one point/goal. Of course one could, rather than counting points or goals, only let it count whether or not it was a victory and set $(i,j) = 1$ if team i beat team j. Because of this interpretation the case of $(0,1)$-matrices is interesting on its own, but now it is time to turn to hillside matrices.

6.4.2 A hidden hillside criterion

Now it is time to move on to the case we are really interested in, namely the hillside form. From our previous work it is clear that if we set all nonzero elements in a matrix to 1, we can check if the graph of this modified matrix is a DAG, and hence if it can be symmetrically permuted to a strict upper triangular matrix.

This would mean that the original matrix (where the nonzero elements are themselves) also can be permuted to a strict upper triangular matrix, but of course we do not necessary have ascending order across rows and descending order down columns (i.e. that the matrix is in hillside form) in all cases. However, we shall see that if this procedure does not yield a hillside form, nothing will.

For the rest of this chapter, we assume that no games were tied. Said differently, this means that d_{ij} and d_{ji} cannot both be 0 for $i \neq j$.

Theorem 6.14. Let $\boldsymbol{D}$ be a point differential matrix. If it has a hidden hillside form, then we can find it by using the permutation one gets from topologically ordering the adjacency graph of $\boldsymbol{D}$.

Proof. Let the $n \times n$-matrix $\boldsymbol{D}$ be a point differential matrix. Assume that is has a hidden hillside form, meaning that there exists a permutation $\boldsymbol{\pi}$ that takes $\boldsymbol{D}$ to hillside form. Let $\boldsymbol{D_\pi}$ be this symmetrically reordered matrix. Now, let $\boldsymbol{D'}$ be $\boldsymbol{D_\pi}$ when we set all nonzero entries to 1. This means that the row sums of $\boldsymbol{D'}$ will be $n-1$, $n-2$, $\ldots$, 0, or that team number $\boldsymbol{\pi}(1)$ beat all other teams, team $\boldsymbol{\pi}(2)$ beat all other teams but $\boldsymbol{\pi}(1)$ and so on.

As $\boldsymbol{D}$ has a hidden hillside form, we know that $\boldsymbol{D}'$ can be permuted to a strict upper triangular matrix. By 6.11, its adjacency graph must then have a topological ordering. This means that we can order the teams $T_1, \ldots, T_n$ in such a way that T_i did not beat T_j when $i > j$. Because all teams played a game and no games ended in ties, this means that team T_1 beat all the other teams, team T_2 beat every team except T_1, and so on.

But this means precisely that $T_i = \boldsymbol{\pi}(i)$, meaning that we can find $\boldsymbol{\pi}$ by ordering the adjacency graph of $\boldsymbol{D}'$ topologically. This means that if D has a hidden hillside form, we get it from topologically ordering the adjacency graph of $\boldsymbol{D}'$. □

Corollary 6.15. If the adjacency graph of a point differential matrix $\boldsymbol{D}$ does not have a topological ordering, then $\boldsymbol{D}$ does not have a hidden hillside form.

We remark that instead of finding the topological ordering of the adjacency graph of a matrix with a hidden hillside form, one could merely sort by descending row sums. However, we stress that this is only valid for the special case where the matrix actually has a hidden hillside form. In the more general case, we cannot be certain that the ranking found when sorting on row sums will yield the minimimum distance from hillside form. Nevertheless, we saw in Section 5.3.5 that sorting on row sums can provide a good initial ranking for matrices without a hidden hillside form.

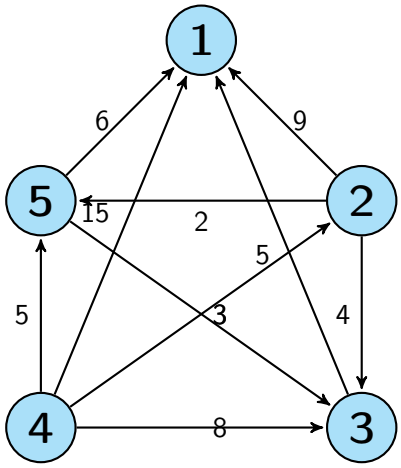
We illustrate Proposition 6.14 by revisiting our example from [PLY12] (Example 5.21). We remark that MATLAB has an in-built function, `graphisdag(G)` (where $\boldsymbol{G}$ is the adjacency matrix of the graph), that checks whether a graph is a DAG or not. The function is boolean, so it returns 1 if the graph of $\boldsymbol{G}$ is a DAG and 0 otherwise.

MATLAB also has a function `graphtopoorder(G)` that takes a dense matrix and returns its topological ordering (if it has any). It will not be done in this thesis, but we also mention that MATLAB can display graphs via the function `biograph`.

Example 6.16. We start with the point differential matrix given as

$$\boldsymbol{D} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 4 & 0 & 2 \\ 5 & 0 & 0 & 0 & 0 \\ 15 & 3 & 8 & 0 & 5 \\ 6 & 0 & 3 & 0 & 0 \end{pmatrix}$$

The adjacency graph of $\boldsymbol{D}$ is the following.



We use two MATLAB functions to solve this; `graphisdag`($\boldsymbol{D}$) and `graphtopoorder`($\boldsymbol{D}$). `graphisdag`($\boldsymbol{D}$) returns 1, which confirms that $\boldsymbol{D}$ is a DAG. We then use `graphtopoorder`($\boldsymbol{D}$) to find an actual topological ordering of the graph.

```
D = [0 0 0 0 0; 9 0 4 0 2; 5 0 0 0 0; 15 3 8 0 5; 6 0 3 0 0];
G = sparse(D);
topord = graphtopoorder(G)
pi = rtopi(topord)
P = permutationmatrix(pi);
D_pi = P'*D*P
```

And we get that

$$\texttt{topord} = \begin{pmatrix} 4 & 2 & 5 & 3 & 1 \end{pmatrix}$$

Since this is a topological ranking, it is of course in our r-notation for rankings; it is a listing of the items from best to worse. Whereas the $\boldsymbol{\pi}$-ranking uses the permutation notation we have used in most of this thesis.

$$\texttt{pi} = \begin{pmatrix} 5 & 2 & 4 & 1 & 3 \end{pmatrix}$$

$$\texttt{D\_pi} = \begin{pmatrix} 0 & 3 & 5 & 8 & 15 \\ 0 & 0 & 2 & 4 & 9 \\ 0 & 0 & 0 & 3 & 6 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We see that by doing this, we end up with the hillside form for the matrix.

This is the same result as we got in Example 5.24, where we sorted on descending row sums. This verifies our previous remark of the fact that for matrices with a hidden hillside form, finding a topological ordering of the adjacency graph of the matrix and sorting by descending row sums are equivalent. □

6.5 Further work

In Theorem 6.14, we found a connection between the hillside form of a matrix and the topological ordering of its adjacency graph. While this only tells us whether or not an *actual* hillside form ($h(D) = 0$) exists, and not *how close* we can get to one, it suggests that algorithms for finding *approximately topological* orderings might be applied to finding approximate hillside forms. We saw another indication of this in Chapter 5, as choosing the initial ranking $\boldsymbol{\pi}$ by sorting the items by descending row sums improved our results.

One possible direction for future work, then, would be studying different notion of approximate topological orderings and seeing whether some correspond to approximate hillside forms. Another direction would be to look for applications outside sports, which has been the main source of our examples. One area in which minimizing $h(D)$ seems more appropriate than minimizing the number of violations in D is elections and voting, as we want our elections to tell the difference between landslide victories and close races.

Appendices

Appendix A

Throughout this thesis I have made many programs. In this appendix follows the codes for these programs. All programs are written for MATLAB, if nothing else is stated.

A.1 Codes for Chapter 4

A.1.1 Finding the cost matrix (Section 4.2)

```
function [C] = costmatrix(D)
% taking the n*n point differential matrix D and returns the cost matrix C
% as defined in the MVR article by Pedings et. al.

n = length(D);
C = zeros(n,n);

for i = 1:n
    for j = 1:n
        contribution = 0;
        for k = 1:n
            if (D(i,k) < D(j,k))
                contribution = contribution +1;
            end
            if (D(k,i) > D(k,j))
                contribution = contribution +1;
            end
        end
        C(i,j) = contribution;
    end
end
```

A.1.2 Convert from decision matrix to ranking (Section 4.2)

```
function rank = findranking(X)
% finds a ranking based on sorting on row sums of decision matrix X
% X is n*n matrix which is a reordering of a upper triangular matrix with
```

```
% all ones.

n = length(X);
rank = zeros(1,n);
colsum = sum(X);

k = 0;
i = 1;
while k < n
    rank(i) = find(colsum==k);
    i = i + 1;
    k = k + 1;
end
```

A.1.3 Convert from ranking to decision matrix (Section 4.2)

```
function X = decisionmatrix(pi)
% returns decision matrix based on ranking pi

n = length(pi);
X = zeros(n,n);
visited = zeros(1,n);

for i=1:n
    for j=1:n
        ind = find(pi==i);
        visited(ind) = 1;
        if visited(j) == 0
            X(ind,j) = 1;
        end
    end
end
```

A.1.4 Counting the number of violations from hillside form for a matrix (Section 4.2)

```
function viols = numberofviols(C, X)
% calculating number of violations from hillside form
% C is cost matrix, X is decision matrix

n = length(C);
viols = 0;

for i=1:n
    for j=1:n
        viols = viols + C(i,j)*X(i,j);
    end
end
```

A.1.5 Implementation of MVR method from [PLY12] (Section 4.3)

This implementation is programmed in OPL for the optimization software CPLEX.

```
int n = ...;
range N = 0..n-1;

int c[N][N] = ...;       /* cost matrix */

dvar boolean x[N][N];    /* boolean will ensure that all values are 0 or 1*/

minimize
sum(i in N, j in N) c[i][j]*x[i][j];

subject to {

 forall(i in N)
   forall(j in N)
     if (i != j) x[i][j] + x[j][i] == 1;

 forall(i in N)
   forall(j in N)
     forall(k in N)
     if (i != j && j != k && i != k) x[i][j] + x[j][k] + x[k][i] <= 2;
}
```

A.2 Codes for Chapter 5

A.2.1 Implementation of Definition 5.3 (Section 5.1)

```
function [distance] = h(D)
% D is a n*n matrix
% h(D) is the distance of D from hillside form

n = length(D);
sum = 0;

for i = 1:n
    for j = 1:n-1
        for k = j+1:n
            row_contribution = max([D(i,j)-D(i,k) 0]);
            column_contribution = max([D(k,i)-D(j,i) 0]);
            sum = sum + row_contribution + column_contribution;
        end
    end
end
distance = sum;
end
```

A.2.2 Converting a ranking from r-notation to π-notation (Section 5.2)

```
function [pi] = rtopi(r)
%converts a ranking r to a ranking pi. I.e. from a list of ranked teams,
%where r(1) is the top ranked team to a ranking where pi(1) is what ranking
%team 1 has.

n = length(r);
pi = zeros(1,n);

for i = 1:n
    j = r(i);
    pi(j) = i;
end
```

A.2.3 Finding the permutation matrix P_π when given a permutation π (Section 5.2)

```
function P = permutationmatrix(pi)
% creates a permutation matrix P based on the permutation on n elements, pi

n = length(pi);
P = zeros(n,n);

for i = 1:n
    P(i, pi(i)) = 1;
end
```

A.2.4 Minimizing distance to hillside form using only transpositions (Section 5.3.1)

```
function [best_D, best_dist, best_pi] = mindistohills(D)
% choosing best transposition, i.e. the one that minimizes distance to
% hillside form, in each iteration. Stops when no improving transposition
% can be found from current ranking.

distance = h(D);
n = length(D);

if distance == 0
    S = sprintf('The matrix D is in hillside form.');
    disp(S)
    best_D = D;
    best_dist = distance;
    pi = linspace(1,n,n);

else
    best_dist = distance;
    pi = linspace(1, n, n);
    best_pi = pi;
    best_D = D;
```

```
    indices = pi;
    possibleindices = nchoosek(indices,2);
    m = length(possibleindices);
    cont = true; % to determine if we continue to look for improvements

    while best_dist ~= 0 && cont == true
        cont = false; %changed to true later if an improvement is found
        pi = best_pi;

        for k = 1:m
            current = possibleindices(k,:);
            i = current(1);
            j = current(2);

            indi = find(pi==i);
            indj = find(pi==j);

            new_pi = pi;
            new_pi(indi) = j;
            new_pi(indj) = i;

            P = permutationmatrix(new_pi);
            new_D = P'*D*P;
            new_dist = h(new_D);
            if new_dist < best_dist
                best_dist = new_dist;
                best_D = new_D;
                best_pi = new_pi;
                cont = true;
            end
        end
    end
end
```

A.2.5 Minimizing distance to hillside form using transpositions and allowing some bad choices (Section 5.3.2)

```
function [best_D, best_dist, best_pi] = mindistohills_alpha(D)
% choosing best transposition in each iteration, when no improvement can be
% found we sometimes (with probability alpha) chose a transposition that
% takes the matrix further from hillside form.

distance = h(D);
n = length(D);
alpha = 0.2;   % the probability that we will make a "bad choice" when no
               % improvment can be found

if distance == 0
    S = sprintf('The matrix D is in hillside form.');
    disp(S)
    best_D = D;
    best_dist = distance;
    best_pi = linspace(1,n,n);

else
    best_dist = distance;
    pi = linspace(1, n, n);
    best_pi = pi;
```

```
22      indices = pi;
23      possibleindices = nchoosek(indices,2);
24      m = length(possibleindices);
25      cont = true; %to determine if we continue to look for improvements
26
27      alpha_pi = zeros(1,n);
28      alpha_D = zeros(n,n);
29      alpha_dist = -1;
30
31      while best_dist ~= 0 && cont == true
32          cont = false; %changed to true later if an improvement is found
33          diff = intmax;
34          pi = best_pi;
35          for k = 1:m
36              current = possibleindices(k,:);
37              i = current(1);
38              j = current(2);
39
40              indi = find(pi==i);
41              indj = find(pi==j);
42
43              new_pi = pi;
44              new_pi(indi) = j;
45              new_pi(indj) = i;
46
47              P = permutationmatrix(new_pi);
48              new_D = P'*D*P;
49              new_dist = h(new_D);
50              if new_dist < best_dist
51                  best_dist = new_dist;
52                  best_D = new_D;
53                  best_pi = new_pi;
54                  cont = true;
55
56              else    % save a "bad choice" perhaps to be used later
57                  if (new_dist - best_dist) < diff
58                      alpha_pi = new_pi;
59                      alpha_D = new_D;
60                      alpha_dist = new_dist;
61                      diff = new_dist - best_dist;
62                  end
63              end
64          end
65
66          if cont == false && best_dist ~= 0
67              badchoice = rand(1,1);
68
69              if (badchoice <= alpha)        % shall we go for a bad choice?
70                  best_dist = alpha_dist;
71                  best_D = alpha_D;
72                  best_pi = alpha_pi;
73                  cont = true;
74              end
75          end
76      end
77  end
```

A.2.6 Minimizing distance to hillside form using transpositions and 3-cycles (Section 5.3.3)

```
function [D_best, dist_best, pi_best] = mindistohills_tri(D)
% minimizing distance to hillside form for D by randomly choosing 3 indices

dist = h(D);
n = length(D);

if dist == 0
    S = sprintf('The matrix D is in hillside form.');
    disp(S)
    D_best = D;
    dist_best = distance;
    pi_best = linspace(1,n,n);

else
    dist_best = dist;
    pi_best = linspace(1, n, n);
    D_best = D;

    tol = 2;          % the number of misses in a row we allow
    count = 0;        % how many misses in a row so far
    while dist_best ~= 0 && count < tol
        count = count + 1;
        sample = randsample(n, 3); % randomly choose 3 indices
        i = sample(1);
        j = sample(2);
        k = sample(3);

        perm = perms(sample);   % all possible permutations of the indices

        pi = pi_best;

        for l = 1:length(perm)  % check all permutations for improvements
            current = perm(l,:);
            pi_test = pi;

            indi = find(pi==i);
            indj = find(pi==j);
            indk = find(pi==k);

            pi_test(indi) = current(1);
            pi_test(indj) = current(2);
            pi_test(indk) = current(3);

            P_t = permutationmatrix(pi_test);
            D_test = P_t'*D*P_t;
            dist_test = h(D_test);

            if dist_test < dist_best  % improvement found, update variables
                dist_best = dist_test;
                D_best = D_test;
                pi_best = pi_test;
                count = 0;      % reset counter
            end
        end
    end
end
```

A.2.7 Brute-force program checking all permutations to minimize distance to hillside form (Section 5.3.4)

```
function [best_D, best_dist, best_pi] = allpermutations(D)
% brute-force. Checks all permutations of n elements to minimize distance
% to hillside form for D.

distance = h(D);
n = length(D);

if distance == 0
    S = sprintf('The matrix D is in hillside form.');
    best_D = D;
    best_dist = distance;
    best_pi = linspace(1, n, n);
    disp(S)
    return

else
    if (n > 10)
        S = sprintf('This will take a long time. Possibly forever.');
        disp(S)
    end

    pi = linspace(1, n, n);
    best_dist = distance;
    best_D = D;
    best_pi = pi;

    permutations = perms(pi);

    for i = 1:length(permutations)
        pi_new = permutations(i,:);
        P = permutationmatrix(pi_new);
        D_new = P'*D*P;
        dist_new = h(D_new);

        if dist_new < best_dist
            best_D = D_new;
            best_pi = pi_new;
            best_dist = dist_new;
            if (dist_new == 0) % stop if matrix is in hillside form
                break
            end
        end
    end
end
```

A.2.8 Finding a better initial guess on the ranking π (Section 5.3.5)

```
function [pi_start] = startpi(D)
% given n*n point differential matrix D, this function aims to find a
% reasonable initial ranking (pi) of the n items by summing the row elements

n = length(D);
```

```
pi_start = zeros(1,n);

rowsum = sum(D,2);
sortedsum = fliplr(sort(rowsum)'); % sorted from largest to smallest

if length(rowsum) == length(unique(rowsum))  % all unique row sums
    for k=1:n
        ksum = rowsum(k);
        rank = find(sortedsum==ksum);
        pi_start(k) = rank;
    end

else % some row sums are identical
    values = unique(rowsum);
    instances = histc(rowsum(:),values);

    for k = 1:length(values)
        if (instances(k) ~= 1)
            noofequalsums = instances(k);
            equalsum = values(k);

            indices = zeros(1,noofequalsums);
            j = 1;
            for i = 1:noofequalsums
                cont = true;
                while (j <= n) && cont
                    if (rowsum(j) == equalsum)
                       indices(i) = j;
                       cont = false;
                    end
                    j = j + 1;
                end
            end

            for r = 1:noofequalsums-1
                i = indices(r);
                j = indices(r+1);

                epsilon = 0.01;
                if (D(i,j) > 0)
                    rowsum(i) = rowsum(i) + r*epsilon;
                else
                    rowsum(j) = rowsum(j) + r*epsilon;
                end
            end
        end
    end

    sortedsum = fliplr(sort(rowsum)'); % sorted from largest to smallest
    for k=1:n
        ksum = rowsum(k);
        rank = find(sortedsum==ksum);
        pi_start(k) = rank;
    end
end
```

Bibliography

[BL06] Kurt Bryan and Tanya Leise. The $\$25,000,000,000$ eigenvector: the linear algebra behind Google. *SIAM Rev.*, 48(3):569–581, 2006.

[BM76] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*. American Elsevier Publishing Co., Inc., New York, 1976.

[Col02] Wesley N. Colley. *Colley's bias-free college football ranking method: The Colley Matrix Explained*. www.colleyrankings.com, 2002.

[Dah12] Geir Dahl. A matrix-based ranking method with application to tennis. *Linear Algebra Appl.*, 437(1):26–36, 2012.

[DM10] Geir Dahl and Harald Minken. A note on permutations and rank aggregation. *Math. Comput. Modelling*, 52(1-2):380–385, 2010.

[Fra67] John B. Fraleigh. *A first course in abstract algebra*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1967.

[Lee] James R. Lee. *DAGs and Topological Ordering*. http://homes.cs.washington.edu/~jrl/421slides/lec5.pdf. Accessed: 03.04.15.

[LM12a] Amy N. Langville and Carl D. Meyer. *Google's PageRank and beyond: the science of search engine rankings*. Princeton University Press, Princeton, NJ, 2012. Paperback edition of the 2006 original.

[LM12b] Amy N. Langville and Carl D. Meyer. *Who's #1? The science of rating and ranking*. Princeton University Press, Princeton, NJ, 2012.

[Mey00] Carl Meyer. *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. With 1 CD-ROM (Windows, Macintosh and UNIX) and a solutions manual (iv+171 pp.).

[PLY12] Kathryn E. Pedings, Amy N. Langville, and Yoshitsugu Yamamoto. A minimum violations ranking method. *Optim. Eng.*, 13(2):349–370, 2012.

[Saa03] Yousef Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.